ONR Workshop on Software Development

June 9-10, 1989

Moscow, Idaho

ONR Workshop on Software Development

June 9-10, 1989

Moscow, Idaho

REPORTS

Structured Approaches for Problems of Network Design and Utilization

Michael R. Fellows, University of Idaho
(Contract N00014-88-K-0456)

Michael A. Langston, Washington State University
(Contract N00014-88-K-0343)

Abstract.

The effectiveness of two general, structured approaches to broad classes of network design and utilization problems is investigated. These are:

(1) an approach to network algorithmic problems based on well-partial orders (wpo's) on sets of combinatorial objects, where the goal is to develop this powerful mathematical perspective into a foundation for practical algorithms and

(2) an approach to symmetric and fault-tolerant interconnection network design and allocation problems employing algebra and coding theory, where the goal is to establish effective design paradigms drawing on established mathematical resources.

The results obtained in these research areas during this period of ONR support and described in this progress report include:

(1a) general methods for overcoming the major difficulties in obtaining practical algorithms for network problems from wpo-based tools,

(1b) improved practical algorithms for some well-known algorithmic problems of networks,

(2a) a host of record-breaking algebraic constructions in the range of engineering significance for the much-studied degree/diameter network construction problem,

(2b) basic results for the planar and planar-symmetric versions of the degree/diameter network construction problem, and

(2c) useful schemes for locally complete data distribution in networks, and general methods for employing algebraic network descriptions to solve this problem.

Background and Objectives.

Networks of many kinds play an increasing role in almost every aspect of modern science and technology, and figure centrally in the forefront of developments in computer science. Problems concerning the design, organization and utilization of networks play a correspondingly important role. For these problems, it is desirable to have useful general tools and methodologies that are *organizing principles*, that is, approaches that can be applied to broad classes of particular problems.

Our research has been centered on the development of two such broad perspectives on network design and algorithmic problems, both of which are based on strong mathematical resources. In the first, we seek to develop the theoretical basis of wpo-based tools so that they might provide a foundation for practical networks algorithms. In the second, we endeavor to demonstrate the effectiveness of algebraic methods for problems of network design. Our research program recognizes and addresses these aspects:

- the emergence of the importance of network problems,
- the need to develop more powerful and well-integrated theoretical perspectives on network problems, and
- the opportunity provided by the recent fundamental mathematical breakthroughs of Robertson and Seymour, and others.

Research Issues, Approaches and Progress.

Our research objectives have been formulated at the level of addressing issues concerning fundamental feasibility of these approaches, rather than that of the creation of software or systems. At this level we have already achieved many of the objectives in our original proposal. We therefore take the opportunity presented by this progress report to articulate an updated set of research objectives for this general line of investigation.

wpo-based methods

Our earlier research has demonstrated the wide applicability of wpo-based tools to many well-known, previously challenging, interesting and useful algorithmic problems. Such applications, however, left unresolved a seemingly enormous gap between these theoretical results and any potential for practical algorithms. The major, virtually unprecedented issues to be faced include:

a. the polynomial-time algorithms guaranteed by the wpo-related theorems are nonconstructively proven only to exist,

b. the algorithms promised involve behemoth constants (towers of 2's of height described by towers of 2's of height ...), and

c. the algorithms promised solve only decision versions of the problems to which the theorems apply.

Confronting these difficulties would seem to constitute an adequate agenda, even for a primarily theoretical investigation! Indeed, so much so that at least one prominent member of the research community predicted that wpo-based techniques would never amount to more than a "mathematical curiosity," at best a signpost for polynomial time.

Yet, by the results we have recently presented at *STOC 89*, these apparent roadblocks have, for the most part, been decisively removed for most of the known applications. Other researchers (for example, Bodlaender) have already begun to augment and extend our techniques for addressing issues b and c.. We have recently obtained even stronger methods for addressing issue a.

A remaining major issue (the size of obstruction sets) is an important new focus in this project. Along with Nancy Kinnersley, a 1989 Ph.D. graduate from Washington State University (whose graduate research was supported by this contract, and who has just accepted a faculty position at the University of Kansas), we have recently completed identification of the 110 obstructions to pathwidth 2. On the basis of this work we have gathered strong evidence to suggest that not all obstructions are created equal, and that for many applications the difficulty can be overcome by employing a reasonably small *approximate* obstruction set that embodies almost all of the structural information about the problem.

Our current objectives include:

- a structure theory for obstruction sets that allows us to understand the possibilities for approximate obstruction sets,
- an understanding of the possibilities for randomized multiple-trial self-reduction algorithms, and how these might interact with approximate obstruction sets,
- the exploration of possible self-reduction algorithms that are designed to work correctly almost always employing only a small set of obstructions, and
- continued research on faster order tests for the important wpo sets.

algebraic methods for network design

The chief advantage of an algebraic approach is that for some applications of large and complex networks that are presently contemplated (for example, in parallel processing) it is natural to use symmetry as an organizing principle. (Thus, for example, in a vertex symmetric network one might just write one message-routing protocol for a node, and translate it into one for all the other nodes.) By symmetry one inevitably means group theory, for which we have well-established mathematical knowledge on which to draw. Algebraic network descriptions have other organizational advantages, including being compact and comprehensible — this can support efficient routing computations, and can be an aid to exploiting the symmetries of a computational problem.

When we began this project two years ago we knew but a handful of largest known constructions of networks of a given degree and diameter by algebraic means (especially Cayley graphs). This has been a much-studied problem and at the time of our proposal almost

all of the largest known constructions for the range of the parameters (degree, diameter) of potential engineering significance had been obtained by various authors, mostly by means of an assortment of graph-theoretic compositional techniques.

Since then we have rewritten the table almost completely, and decisively demonstrated the power of an algebraic approach to this well-known network design problem. This was the principle initial objective for this topic in our original proposal. Along with C. S. Jagadish, a graduate student at the University of Idaho supported under this award, we have made basic advances on the planar and planar vertex symmetric variations on this problem.

Another problem that our research has addressed is that of devising efficient schemes for storing partial copies of a database at the nodes of a network so that each node has a complete copy of the entire database in its immediate neighborhood. Algebraically described networks support efficient algebraic solutions to this problem; we have obtained asymptotically optimal solutions for the hypercubes, in the process settling in the affirmative a conjecture in coding theory due to Cohen.

Our current objectives include:

- to improve the largest known degree/diameter constructions for small diameter values by employing Cayley coset graphs,
- to explore an algebraic approach to the degree/broadcast diameter construction problem,
- to explore ways of using the algebraic descriptions for classes of good constructions to efficiently solve routing and deadlock problems, and
- to continue our exploration of locally complete data distribution schemes in algebraically described networks, seeking especially improved schemes for small dimension hypercubes and other network families important in parallel processing.

Research Directions.

wpo-based methods

At this point, this exciting research frontier appears to be significantly undermanned relative to its potential, perhaps in part due to the initial bad publicity surrounding the difficulties a through c described above, and the daunting nature of the founding mathematical results (the completed proof of the main Robertson-Seymour theorems concerning the minor order of graphs comprises approximately 1600 journal pages). It does seem remarkable that the many interesting consequences of these deep theorems, and the basic questions for computer science that they raise, were essentially absent from the major conferences in theoretical computer science for a period of almost 5 years after their announcement. It is quite difficult to imagine a similar intellectual sociology occurring in, say, physics or chemistry.

There are a great many leads worth pursuing. The more theoretical aspects of the theory and its applications to computer science are beginning to gain researchers and momentum,

perhaps for the most part recruited from mathematics. As we have reported, however, most of the major issues for a host of potentially interesting applications have been overcome. The time is thus ripe for significant experimental exploration with implemented algorithms, but there is as of yet almost no such research activity, suggesting that:

- the methods for computing and mechanically verifying obstruction sets that we have recently developed should be tried out experimentally for some small applications,
- a library of implementations of the best known minor tests should be assembled, and
- the "learning algorithm" constructivization should be explored experimentally for some small examples.

Note that an attractive feature of wpo-based tools is their modular nature. For a given application, the relevant order tests can be performed trivially in parallel, and a single order test might be useful in many different applications, and can thus reside in a universal library. Perhaps this library itself can be to some extent mechanically generated.

algebraic methods for network design

The major direction that awaits exploration based on our results is whether the algebraic constructions that we have identified can be fully developed into, for example, superior alternatives to the hypercubes for parallel processing networks. This necessarily involves consideration of many more aspects than merely degree/diameter properties. We have recently begun to explore a number of these aspects in collaboration with Vance Faber of the Los Alamos National Laboratory. The intuition behind our work is that much of what makes the hypercube (and most other proposed network topologies) attractive is, essentially, the fact that it has an easily manipulable algebraic description.

A Grand Challenge.

Wpo-based tools should not be perceived as exotic or special, but rather as a kind of "generalized" brute force. Interesting families of combinatorial objects are often closed under local operations by which a partial ordering of the objects can be defined. The basic rule of thumb is that in *some* sufficiently restricted setting this partial order is a wpo that supports wpo-based complexity tools.

For some sets of operations (e.g., those defining the minor and immersion orders) the setting is simply all the objects, while for others the best setting available is more restrictive. For example, the operations (1) remove a subdivision and (2) take a subgraph (which define the topological order) do not yield a well-partial order in general, but this does constitute an wpo in the setting: all graphs that do not contain k disjoint cycles. (We have recently shown that for this wpo all order tests can be done in linear time.)

The Hilbert-sized problem is to provide a comprehensive explanation of what sets of operations on what sets of objects yield wpo-based complexity tools.

Research Transitions.

At this time, there is an enormous cultural and intellectual gap in many problem domains between researchers who explore algorithmic problems of graphs and networks theoretically, and engineering practitioners who are forced to "hack away" at problems that must be dealt with immediately, somehow. What realistically can be done about this situation is not completely clear.

Technological Impacts.

Our research is primarily theoretical in nature and thus is not significantly impacted by or awaiting new developments in hardware and software tools. One rather minor exception to this statement concerns our recent computational exploration of some Cayley networks having in the range of 0.5 to 10 million nodes. A problem we encountered concerned finding a machine with sufficient fast memory.

Societal Issues.

We seem to have entered an era in which the great majority of the most able and motivated graduate students are foreign students. We believe this is a partial reflection of the fact that graduate study is so very poorly supported, despite the often major contributions to the research frontier, even in the short run, that are due to energetic and talented students. *A truly vigorous program for research-oriented graduate study would be a highly leveraged investment in this nation's security and quality of life.*

Recommendations to Funding Agencies.

Science funding sometimes seems to us to be rather akin to the child's toy "chinese handcuffs," in that the more that funding agencies strain for short-term payoffs and objectives, the less progress will be made in the long haul. We believe that it should be the objective of science funding agencies to provide broad-based support of science and facilitate the transfer of basic science towards classrooms, industry, defense and other applications. It also seems to us that basic science is most efficiently supported by grants to individual researchers or small groups, with funding decisions determined heavily by scientific peer review, or some process that is highly respectful of a science's critical self-evaluation.

ALGORITHMS BASED ON GRAPH DECOMPOSITION

MATTHIAS F.M. STALLMANN*

Abstract. This progress report describes research on dynamic programming algorithms and related issues. Applications include problems in parameter estimation in PERT networks, network reliability, algorithms for NP-hard graph problems, hypercube embedding, and graph coloring problems that arise in code optimization and via minimization for VLSI.

1. Background. This research project has evolved into one whose central issue is dynamic programming. It was not intended that way originally, nor did I approach it as someone who was interested in becoming an expert on dynamic programming. I should clarify at the outset that my use of the phrase "dynamic programming" is not a technical term, as it might be in certain circles of the operations research community. It is rather a broad, but vaguely defined, framework for algorithm design in which solutions to large instances of a problem are built up from solutions of smaller instances of the same problem. Most of the standard textbooks on algorithm design recognize the importance of dynamic programming as an algorithm design technique (see e.g. [2]). Because of its utility in solving practical problems, it could be argued that dynamic programming is the most important algorithm design technique. Dynamic programming algorithms are easy to formulate and easy to implement and are often very efficient. Robustness is the main selling point of dynamic programming algorithms – it is usually easy to incorporate additional constraints or to adapt from a cardinality problem to a weighted problem.

I came around to this view only recently, having been schooled in all the fine points of efficient data structures, graph searching, and augmenting path algorithms (all of which have been extremely useful). When I came to NCSU, I decided, since I was now a theoretician in a place that put great emphasis on practical applications, to do a lot of listening to colleagues and students, and to glean from them the areas in which theory might prove useful. I kept a particular lookout for problems that were suspected to be intractable (NP-hard or worse), but had no natural structure to suggest an NP-hardness proof, hoping to find matroid parity problems lurking in them (since my thesis was about matroid parity).

I haven't stumbled across any matroids, but I have become better at proving NP-completeness results and at devising dynamic programming algorithms for problems that are not NP-hard. The problem which is the central focus of this report, directed acyclic graph reduction, has an interesting story behind it that illustrates how I have chosen some of the problems I've worked on, or rather how they have chosen me.

Some time in late 1986 or early 1987, Salah Elmaghraby asked me if there was an efficient way to enumerate all the IG's (subgraphs homeomorphic from the dag pictured in Figure 1) of a dag. He was in the process of writing a survey paper on

* Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206

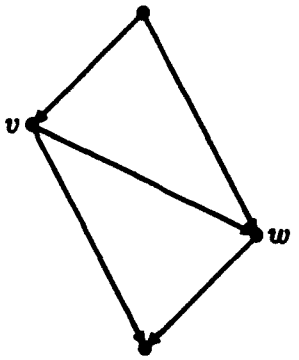

FIG. 1. *The interdictive graph (IG).*

parameter estimation in PERT networks, and getting rid of IG's appeared to be a central issue. I remembered a result about subgraph homeomorphism, that it could be done in polynomial time for dags [19], and pointed out that this could be used, at the very least, to list all pairs of vertices that were in positions v and w of an IG (see Figure 1). Then I made the mistake of asking why he wanted to do such a thing. The real problem he was after was to minimize the number of node reductions required to reduce a dag to a single edge (see Section 4 for a full description). He felt sure that the problem was NP-hard (maybe I could help him prove it) and the enumeration of the IG's would reduce it to vertex cover, a more manageable problem. Some weeks later I made the observation that the vertex cover problem he was hoping to reduce to (a) did not model the original problem exactly, and (b) when it did, the graph defining the vertex cover problem was transitive, which made me suspicious that there might be a polynomial time algorithm for the original problem. After more than a year of working through examples and counterexamples of a variety of different conjectures, and enlisting the aid of a colleague from Duke (Wolfgang Bein, who knew a lot more about series-parallel dags than I did), I developed a polynomial time algorithm for the original problem, reducing it to vertex cover in a transitive dag [48]. It took me almost another year to fully understand the applications, and I'm still not convinced that the right quantity is being optimized for some of them.

Similar stories are behind my current work on via minimization in VLSI and hypercube embedding, except that in each of these cases it was a student, rather than a colleague from another department, who provided my original contact with the problem.

2. Research Objectives. Many graph problems arising in practical applications are NP-hard when stated as general graph problems, but may in fact be easy when the special structure of graphs arising in the application is considered. Johnson [25] gives a survey of results applying to many special classes of graphs. Many of the polynomial-

time results are due to dynamic programming algorithms that take advantage of some form of decomposition for the graphs in question. The long-term objective of this research is to develop general techniques for obtaining polynomial-time algorithms for special cases of NP-hard problems, particularly graph problems. Several shorter term objectives flow naturally out of previous work. One is to extend linear-time dynamic programming algorithms to larger classes of graphs than the ones in which they are currently used (see, for example, [4]). Another is develop efficient algorithms for a variety of problems on graphs that are "nearly" series-parallel (see [48] for one approach). Finally, to render the long-term objective more manageable, it is necessary to find linear-time or log-space reductions among problems known to be in P. The importance of this is discussed further in the following section.

3. Research Issues. Dynamic programming has been used successfully as a general technique for obtaining efficient algorithms for problems on special classes of graphs, and for obtaining polynomial-time algorithms in general. For example, Prim's algorithm for minimum spanning trees and Dijkstra's algorithm for shortest paths can be construed as special cases of dynamic programming. Examples using special classes of graphs are legion (see [25] for a partial survey). Unifying approaches, such as that taken by Bern et al [4] are particularly helpful in the design of efficient and elegant algorithms. Other than dynamic programming, the main general techniques for obtaining polynomial-time algorithms are greedy, divide and conquer, successive augmentation, and linear programming. Greedy and divide and conquer can be viewed as special cases of dynamic programming. The other two techniques are more specialized, but in some cases, for example network flows, are being supplanted by simpler techniques. This suggests that dynamic programming is a universal technique for obtaining polynomial-time algorithms.

A more general but related issue is the following. When tackling a new computational problem, a researcher is often focused on the question, "can I design an algorithm for this problem?" This leads to a proliferation of algorithms, some essentially identical to others already published. One of the goals of theoretical research in the area of algorithms is to propose general frameworks for algorithm design to go along with general categories of problems. The important questions, if one takes the theoretical point of view, are

1. What other well-know problems are at least as easy (hard) as the new problem?
2. Is it possible to design an algorithm for the new problem within a given restricted framework?

The first of these is unquestionably useful to both theory and practice. Connections among different problems stimulate progress on all fronts and allow researchers to determine when progress is unlikely. Connections also establish a set of central problems on which researchers can focus their efforts. For example, matrix multiplication is the dominating component in the complexity of many other problems.

The second question at first glance appears to restrict the search for algorithms unnecessarily. Why should we insist on a dynamic programming algorithm, for example, when any algorithm will do? On the practical side, implementation effort is reduced

for algorithms that conform to a specific framework. Also the search for solutions to new problems becomes more focused if it is restricted to frameworks that have already proved successful for similar problems. On the theory side, restricted frameworks put lower bounds within reach, lower bounds that say any algorithm for problem P within framework F takes time $\Omega(f(n))$. Such a lower bound often leads to the discovery of a faster algorithm when the restrictions of F are relaxed. In any case, either a lower bound or an upper bound will lead to new insights about the problem and enhance our understanding of the framework. An important issue, particularly in the case of a framework as general as dynamic programming, is how to define the framework precisely. The definition must be restrictive enough to allow for the possibility of lower bounds (this may be difficult in the case of dynamic programming), but flexible enough to admit a broad range of algorithms.

Another issue is the classification of NP-hard graph problems by relative difficulty using special classes as a guide. For example, hypercube embedding, because it is NP-hard even for trees [54], may be regarded as harder than dominating set, which is easy for trees. In turn, dominating set is NP-hard for chordal graphs, and thus may be regarded as harder than vertex cover (see [25]). Any nested sequence of graph classes induces a linear order on classes of NP-complete graph problems. Two problems are in the same class if they are NP-complete for the same classes of graphs. Aside from being an interesting intellectual exercise, filling in the details of such a classification scheme may lead to insights about the structure of NP-complete graph problems on the theoretical side and better heuristics and algorithms for solving them on the practical side.

4. Approaches. In its most general form, dynamic programming is a collection of rules for determining the solution to an instance of an optimization problem. The optimum solution is either computed directly, if the instance is small enough, or as a simple function of smaller instances of the same problem. If the number of distinct smaller instances that need to be considered during the computation of a large instance is bounded by a polynomial in the size of the large instance, a polynomial time algorithm typically results. In the case of graph problems, the smaller instances almost always involve subgraphs of the original graph. Many classes of graphs can be defined in terms of composition rules that combine smaller subgraphs into larger ones. Bern et al [4] present a general theory for the interaction among composition rules and dynamic programming algorithms.

Where composition rules lead to polynomial-time algorithms on special classes of graphs, it is sometimes possible to extend the rules to general graphs and obtain algorithms that are exponential only in a parameter that measures how often the original rules had to be extended. For example, any two-terminal series-parallel dag (directed acyclic graph) can be defined in terms of a unique (up to associativity of the operators) decomposition tree. Every edge $e = (v, w)$ is a leaf of the tree and has terminals v and w. An interior node representing *parallel composition* joins two subtrees T_1 and T_2, each having terminals v and w, into a single tree $T_1 + T_2$. If G_1 is the dag represented by T_1 and G_2 the dag represented by T_2, the dag represented by $T_1 + T_2$ is $G_1 \cup G_2$ (the two

are joined only at vertices v and w). A *series composition* joins T_1 with terminals u, v and T_2 with terminals v, w into $T_1 \cdot T_2$. Again, the dag represented by $T_1 \cdot T_2$ is $G_1 \cup G_2$, except now the two subgraphs only share vertex v. The source of the new dag is u and its sink is w.

When traversed bottom up, a decomposition tree for a series parallel dag G corresponds to a reduction of G to a single edge. At a parallel composition node with terminals v and w, we do a *parallel reduction*, replacing two edges e, f joining v to w by a single edge $g = (v, w)$. At a series composition node which joins two subgraphs at vertex v, we do a *series reduction* at v. This occurs when $e = (u, v)$ is the unique edge into v and $f = (v, w)$ is the unique edge out of v: e and f are replaced by $g = (u, w)$.

Algorithms for problems such as vertex cover can be formulated in terms of the reduction sequence. For every edge $e = (v, w)$ occurring during the reduction (e may be an edge of the original dag or it may represent a whole subgraph which is joined to the rest of the graph at the two endpoints of the edge), let $VC(e, b_v b_w)$ be the cardinality of the minimum vertex cover for the subgraph represented by e, given that $x = v$ or w is included (excluded) in the cover only if $b_x = 1$ ($b_x = 0$). If e is an edge of G, then $VC(e, 00) = \infty$ (no cover can exclude both endpoints of the edge), $VC(e, 01) = VC(e, 10) = 1$, and $VC(e, 11) = 2$. If g is the edge resulting from a parallel reduction of e and f, then

$$VC(g, b_v b_w) = VC(e, b_v b_w) + VC(f, b_v b_w) - (b_v + b_w)$$

(note that v or w is counted twice if it is included in both covers). If g results from a series reduction of e and f, then

$$VC(g, b_u b_w) = \min\{VC(e, b_u 0) + VC(f, 0 b_w), VC(e, b_u 1) + VC(f, 1 b_w) - 1\}.$$

The cardinality of the minimum vertex cover of G, a two-terminal series parallel dag, can be computed by reducing G to a single edge $e = (s, t)$ and considering the minimum value among the $VC(e, b_s b_t)$.

An arbitrary two-terminal dag can be reduced to a single edge if one additional operation is added to our repertoire. A *node reduction* at v occurs when v has indegree or outdegree 1 (a node reduction is a generalization of a series reduction). Suppose v has indegree 1 and let $e = (u, v)$ be the unique edge into v. Let $f_1 = (v, w_1), \ldots, f_k = (v, w_k)$ be the edges out of v. Replace $\{e, f_1, \ldots, f_k\}$ by $\{g_1, \ldots, g_k\}$, where $g_i = (u, w_i)$. The case where v has out-degree 1 is symmetric ($e = (v, w)$, $f_i = (u_i, v)$, $g_i = (u_i, w)$).

For convenience, let $G \circ v$ denote the result of a node reduction with respect to node v, and let $[G]$ denote the graph that results when all possible series and parallel and parallel reductions have been applied to G (this is well defined because series and parallel reductions obey the Church-Rosser property: the order in which reductions are applied does not affect the final outcome [53]). A dag G is said to be *irreducible* if $[G] = G$.

Let $\mu(G)$, the *reduction complexity* of G, be the minimum number of node reductions which are sufficient (along with series and parallel reductions) to reduce G to a single edge. More precisely, $\mu(G)$ is the smallest c for which there exists a sequence $v_1, \ldots, v_c$

such that $[\cdots[[[G] \circ v_1] \circ v_2] \cdots \circ v_c]$ is a single edge. The sequence $v_1, \ldots, v_c$ is called a ***reduction sequence***. Last year, Wolfgang Bein and I developed a polynomial-time algorithm for computing $\mu(G)$ [48]. The problem of computing $\mu(G)$ is reduced to a problem of finding a minimum vertex cover in a transitive auxiliary graph $C(G)$.

Suppose $v_1, \ldots, v_c$ is a reduction sequence for G and let $\hat{V}$ be an arbitrary subset of $\{v_1, \ldots, v_c\}$. Using a minor modification of the vertex cover algorithm described above, we can compute the cardinality of the minimum vertex cover of G under the restriction that every vertex of $\hat{V}$ must be included in the cover and every vertex of $\{v_1, \ldots, v_c\} - \hat{V}$ must be excluded. Let $g_1, \ldots, g_k$ be the edges resulting from a node reduction at v, where v has indegree 1. If $v \in \hat{V}$, then $VC(g_i, b_u b_{w_i}) = VC(e, b_u 1) + VC(f_i, 1 b_{w_i}) - 1$; otherwise $VC(g_i, b_u b_{w_i}) = VC(e, b_u 0) + VC(f_i, 0 b_{w_i})$. Since every possible subset of $\{v_1, \ldots, v_c\}$ must be considered as a choice for $\hat{V}$, the result is an $O(m2^c)$ algorithm for computing vertex cover in a two-terminal dag.

Problems, such as vertex cover, independent set, dominating set, clique, and coloring, which can be solved by the method described above, are most often formulated on undirected graphs. A biconnected undirected graph can be turned into a two-terminal dag by means of an *st-numbering*, a numbering of the vertices in which vertex 1 is adjacent to vertex n and each other vertex has at least one lower numbered and one higher numbered neighbor [33,17]. Since solutions to the graph problems listed above can be computed separately for each biconnected component and then combined, a natural definition for the *reduction complexity* of an undirected graph G is the maximum over all biconnected components C of G of the minimum over all dags C' resulting from st-numberings of C of $\mu(C)$. Undirected graphs of complexity 0 are exactly the undirected series-parallel graphs. Recognition of undirected graphs of any fixed complexity c appears to be difficult (there are simple examples that show c to be dependent on the st-numbering chosen, even if the numbering of 1 and n is fixed), but is known to be in P by non-constructive methods, using the observation that the complexity of an undirected graph never increases when an edge is deleted or contracted [40,41,18]. An interesting avenue of research is to find specific algorithms for recognizing undirected graphs of complexity c. It is also possible that the recognition problem is NP-complete if c is part of the input.

Several practical applications of reduction complexity are discussed in the next section. A primary activity of this research in the next year will be to learn about dag reduction by applying it to a wide variety of problems of varying difficulty. In some cases, we may find that, whereas dag reduction is not a useful algorithmic technique, our attempts to apply it have yielded non-trivial insights about the problem in question.

As I have gained experience with dynamic programming, I have also been drawn to other problems and other special classes of graphs where dynamic programming is or could be a key factor.

One of these is hypercube embedding: given a graph $G = (V, E)$ and an integer k, find a one-to-one mapping $h : V \rightarrow \{0, \ldots, 2^k - 1\}$ that minimizes either $\max_{\{v,w\} \in E}\{d(h(v), h(w))\}$, the *dilation*, or $\left(\sum_{\{v,w\} \in E}\{d(h(v), h(w))\}\right) / |E|$, the *average dilation*, where $d(i, j)$ is the Hamming distance between i and j, i.e. the number of

bits that differ in their binary representations. Minimizing dilation is important when synchronous parallel algorithms are mapped to the hypercube architecture (in this case G represents the communication structure of the algorithm). Average dilation, particularly a weighted average, may be more important in the case of asynchronous algorithms. Average dilation is also important to coding theory, where the vertices of G represent words to be coded and edges are between words that have similar meanings (and can afford to be given similar codes). Both problems are NP-hard even for trees [54], but the special case of binary (or other bounded degree trees) is still open.

Previous work on average dilation has been primarily experimental, concentrating on various heuristics [6,32,10,16]. Theoretical work on dilation has used separators to obtain low dilation embeddings for various special cases [55,36,5]. An important outstanding conjecture is that any binary tree can be embedded with dilation 2. The best result obtained so far yields dilation 5 embeddings for arbitrary binary trees [36]. We are considering three problems: (1) minimizing dilation, (2) minimizing average dilation, and (3) minimizing the number of edges that have to be deleted in order to achieve an embedding with fixed dilation d. All three are NP-hard for trees, but open for fixed-degree trees. The goals of the hypercube research are:

1. Settle the status of problems (1), (2), and (3) for binary trees (NP-hard or polynomial).
2. Improve existing bounds for dilation in binary trees.
3. Develop heuristics with provably good performance for each of the three problems.
4. Develop good strategies for solving each of the three problems and compare them experimentally.

The other problem is what I call *maximum node coloring*: given a graph $G = (V, E)$ and an integer k, find a maximum cardinality (or weight) subset V' of V such that the subgraph of G induced by V' can be colored with at most k colors. This problem has application to code optimization (register allocation) [1,8,13] and to unconstrained via minimization [23]. In the case of code optimization, the vertices of G represent variables and an edge means that the two variables cannot be stored in the same register. k is the number of registers available – the uncolored variables must be stored in memory. In via minimization G is a circle graph whose vertices represent wires to be routed, an edge means that the two wires cross and cannot be routed on the same layer, k is the number of layers available – the uncolored wires must be routed by means of vias, or contact cuts from one layer to another. The maximum node coloring problem is known to be NP-hard for planar graphs [34,56] and for circle graphs [39,42], in both cases even if $k = 2$. It appears to be solvable for interval graphs and permutation graphs, using dynamic programming, if k is fixed [45]. Both classes are important because exact solutions for graph coloring on them are used to obtain good heuristics for circle graph coloring [50,52] (note: circle graph coloring is NP-hard when $k \geq 4$, but still open when $k = 3$). Interval graphs occur in the register allocation problem when it is restricted to straight-line segments of code. I intend to learn more about maximum node coloring by attempting to find polynomial algorithms or NP-completeness results on other classes

of graphs.

An overview of the approaches currently being considered in this research is given by the following list.

1. Look at dynamic programming algorithms that rely on composition rules for special classes of graphs and extend these to more general graphs, yielding algorithms that are exponential only in some measure of how nearly the graph belongs to the special class.
2. Look for dynamic programming algorithms for problems that are not known to be solvable in polynomial time, in some cases settling for algorithms that are exponential only in a parameter that is not likely to be large in practice.
3. Look for dynamic programming algorithms for problems that are known to solvable in polynomial time, but by methods that are not related to dynamic programming. Examples include matching and network flows (see Section 7 for more details).

5. Progress. Progress to date has been in three main application areas. First there has been much work on the details and proofs of the Stallmann-Bein result on node reduction and its application to problems in operations research. Some of this work is joint work with Jerzy Kamburowski, who independently formulated an algorithm for computing the reduction complexity of a dag. I have been able to show that his algorithm is essentially equivalent to ours and this has simplified some of the proofs in our paper. We have also been able to reduce the time bound of the algorithm from $O(n^3)$ to $O(n^{2.5})$. I am learning about the various OR applications, and expect to be able to suggest improvements in their formulation – that is, to interpret them in a more general framework. This work is described in more detail below, after a brief description of progress on two other fronts.

In the area of hypercube embedding, a student, Woei-Kae Chen, and I have attempted to use dynamic programming to obtain dilation 2 embeddings of binary trees in hypercubes. The only success we have had so far is an algorithm that embeds binary trees with an asymptotic *average dilation* of $2 - \frac{\log n}{n}$ [12]. This is not too promising because many simple heuristics routinely obtain average dilations close to 1 in experimental studies [9]. For this particular project, we have used both theoretical and experimental approaches to attack the problem. Many of our failed dynamic programming algorithms were based on conjectures whose smallest counterexample had 32 or more nodes (in one case, only one counterexample of size 32 existed), so computational trials based on exhaustive search were a valuable resource. We are currently comparing a variety of heuristics, including simulated annealing, in an experimental study [11]. Our methodology is similar to that of Johnson et al [26].

My work with Tom Hughes and Wentai Liu in the area of via minimization has led to a submitted paper [49] and an NSF proposal. I anticipate a lot of future work on algorithms and heuristics for various subproblems related to UVM-based routing. Some preliminary work on max node coloring in special classes that are subclasses of circle graphs is likely to be useful in developing efficient heuristics. The work on UVM based routing has also led me to some ideas on a constrained planar embedding problem that

appears to be difficult, but many special cases can be solved in polynomial time [46]. Dynamic programming may be a factor in solving the general version of this problem, if an efficient algorithm exists.

I am in the process of writing a paper with Salah E. Elmaghraby and Jerzy Kamburowski on applications of dag reduction [15]. The paper features some simplifications of the definitions and proofs in the Stallmann-Bein paper as well as descriptions of several applications to problems in operations research. Consider, for example, the following three problems on a two-terminal acyclic network $G = (V, E)$ in which the weight of each edge e is a random variable X_e governed by a probability distribution function F_e:

1. computation of the pdf of project completion time, where the network is interpreted as a PERT network (activity-on-arc) and edge weights represent durations of the activities,
2. computation of the pdf of the length of the shortest route from source to sink, where the network is a transportation network and weights represent travel time, and
3. computation of the reliability of the network given that weights are either 0 or 1 (0 if the edge fails, 1 if it remains intact).

Let $\mathcal{P}$ be the set of all source-sink paths in G. Then the solutions to the three problems may be formulated as follows:

1. $T = \max_{P \in \mathcal{P}} \sum_{e \in P} X_e$
2. $L = \min_{P \in \mathcal{P}} \sum_{e \in P} X_e$
3. $R = \max_{P \in \mathcal{P}} \prod_{e \in P} X_e$

The pdf's of random variables T, L, and R can be computed using dag reduction as follows (we compute the pdf associated with each edge introduced during the reduction). Suppose g is the parallel reduction of e and f. Then, in the case of T and R, $X_g = \max\{X_e, X_f\}$ and $F_g(x) = F_e(x)F_f(x)$. In the case of L, $X_g = \min\{X_e, X_f\}$ and $F_g(x) = 1-(1-F_e(x))(1-F_f(x))$. Now let g be the result of a series reduction of e and f. In the case of T and L, $X_g = X_e + X_f$ and $F_g(x) = F_e \cdot F_f(x) = \int_0^\infty F_e(x-y)dF_f(y)$. In the case of R, $X_g = \min\{X_e, X_f\}$ and $F_g(x) = 1 - (1 - F_e(x))(1 - F_f(x))$.

Node reductions are complicated by the fact that the pdf's of the g_i's are not independent. Let $g_1, \ldots, g_k$ be the edges resulting from a node reduction (where v has indegree 1 – the other case is symmetric) of e and $f_1, \ldots, f_k$. We reflect the dependence among the F_{g_i} by computing for each g_i the conditional pdf $F'_{g_i}(x;t)$, which is $F_{g_i}(x)$ given that the value of X_e is fixed at t. The final result for the network must then be integrated over all possible values of t with respect to $dF_e(t)$.

If the number of distinct values taken on by edge weights is a fixed constant U, the total time required to compute $Prob(Q \leq x)$ for a given x, where Q is one of T, L, or R, is $O(mU^c)$, where c is the reduction complexity of G (in the case of R, $U = 2$; a simpler formulation is given in [48]). It appears that for these problems, as well as for many other problems, the solution on *autonomous subnetworks*, essentially triply connected components of the dag, can be computed independently. Therefore, the correct measure of reduction complexity should be the maximum over all subnetworks of the complexity

defined above. This would be equally easy to compute; in fact, autonomous subnetworks correspond to connected components of the auxiliary graph [48].

The above results for PERT networks rely on an activity-on-arc representation, which is the one used most often in computations. However, for formulating problems, the activity-on-node representation is more natural, since it does not require the introduction of *dummy activities*. The problem of minimizing the number of dummy activities when translating from activity-on-node to activity-on-arc representation is NP-hard [30]. Kamburowski has conjectured that there is a polynomial algorithm for translating an activity-on-node network G to an activity-on-arc network G' that has minimum reduction complexity among all such G'. This would allow us to extend our algorithmic results to scheduling problems with precedence constraints (precedence constraints are usually represented by activity-on-node networks). A student, David Michael, is working on this conjecture for his PhD thesis.

Other applications of reduction complexity include conditional Monte Carlo sampling, bounds on expected values of random variables in stochastic networks, and dynamic programming approaches to optimal resource allocation in PERT networks.

Another area of progress has been in improving time bounds for computing the auxiliary graph $C(G)$, and for computing a minimum vertex cover of a transitive graph. The latter has been shown to be equivalent to bipartite matching [27], improving the time bound from $O(n^3)$ to $O(n^{2.5})$. A dag whose transitive closure is $C(G)$ can be computed in time $O(n^2)$ [47]. It is also easy to show that computing $C(G)$ is at least as hard as computing the transitive closure of G, and that computing $\mu(G)$ is at least as hard as bipartite matching, hence all our bounds are tight, barring any improvements in the time bounds for transitive closure or matching.

6. Research Directions. The primary research directions suggested by this project have already been discussed in Section 4. One important issue that has been neglected so far, however, is that of efficient parallel algorithms. Dynamic programming algorithms based on tree structured decomposition schemes, such as those for series-parallel graphs, can be efficiently parallelized using tree contraction [35,22]. Efficient dynamic programming algorithms for systolic arrays and meshes have also been proposed [29]. These observations suggest two important lines of research.

1. To what extent can dynamic programming schemes that yield sequential polynomial-time algorithms be adapted to parallel models, such as PRAM, systolic linear array, or mesh?
2. Are there universal schemes, like dynamic programming, that lead to efficient parallel algorithms for problems on special classes of graphs, or for general graphs?

The most promising approach is to attempt to generalize existing parallel algorithms for problems that have been solved sequentially by dynamic programming. Any patterns that emerge should be applied to problems for which no efficient parallel algorithms are known. This is just a general idea and is only being pursued in a limited way by this project. I refer here to the model for on-line systolic graph algorithms mentioned in the proposal: a model of computation for graph problems in which the processing

unit is able to store a small fixed number of data items per vertex and is able to read the edges of the graph sequentially as many times as is required to solve the problem (each reading of the edges is called a *pass*). The processor itself may be a linear array, a mesh, or a random access machine (either sequential or parallel). It appears that while the capabilities of the processor may affect the running time per pass, the total number of passes is not affected. Problems such as connectivity and biconnectivity can be solved in O(1) passes, in linear time on an array and in almost linear time on a sequential machine (see [43,44,51]). Other problems that might be solvable in O(1) passes (though no algorithm is known at this time) are finding a minimum spanning tree and finding a shortest path between two specific vertices. The only progress I can report is not the result of my work: an on-line systolic algorithm for minimum spanning trees by Huang [24].

Since I'm relatively new to this work, I'm not sure I can safely suggest any approaches that should not be pursued. Most of the approaches I've suggested are brute-force, seat-of-the-pants type research, requiring few deep mathematical results. This is not to suggest that the work requires no mathematical background or sophistication. But, as is often the case with combinatorial methods, the mathematics gets made up as you go along, mathematical ideas emerge as the essence of the problem is more clearly understood. It is difficult to predict in advance which mathematical tools will be required for the task at hand. Graph theory and combinatorics are general realms in which to look for tools, but I've found that it's difficult to keep up with all recent results in these areas that may be relevant to even one computer science problem. Communication with people who are knowledgeable in combinatorics and graph theory is essential when the underlying combinatorial problem has been abstracted out of an applied problem.

7. Grand Challenge. Two of the hardest problems that are known to be solvable in polynomial time are graph matching and network flows. Until recently, all known efficient algorithms for both problems have used some form of augmenting path search. The advent of "preflow-push" algorithms for network flows [20] suggests that the flow problem is amenable to more localized strategies. New algorithms for matching have been motivated by parallel models of computation and have centered on the computation of symbolic determinants using randomized algorithms [28]. Deterministic algorithms based on determinants exist for special classes such as planar graphs (follows directly from ideas outlined in [3]) and strongly chordal graphs [14]. However, for planar graphs it is not known how to compute the actual matching deterministically; only the problem of determining whether a perfect matching exists has been solved.

I believe that dag reduction may be helpful in the development of alternate algorithms for network flows and matching. The grand challenge is to find algorithms that are either simpler, more efficient, or more easily parallelizable than existing algorithms for network flows and matching (note: the network flow problem is known to be log-space complete for P [21], hence a polylog-time parallel algorithm is unlikely).

The flow problem for dags is at least as hard as that for general directed graphs [38] (it would be an interesting exercise to try to extend this construction to other

problems, such as min cost flows or network reliability). For series-parallel dags, the network flow problem can be solved in linear time using the following algorithm to compute $MF(e)$, the maximum flow for the subgraph represented by the edge e, for every edge e that occurs during a reduction of G. If g is the result of a parallel reduction of e and f, $MF(g) = MF(e) + MF(f)$; in the case of a series reduction, $MF(g) = \min\{MF(e), MF(f)\}$.

I'm currently looking at how these ideas can be extended to more general dags. One key idea is that it's possible to reduce an arbitrary dag using only node reductions on nodes with indegree 1 (such a reduction sequence may not be minimum, but that's not important to what follows). In a node reduction of v, where $e = (u, v)$ is the unique edge into v, the key issue is how to distribute the capacity of e among the g_i's that result from the node reduction. In the preflow-push model this translates into a decision about how to distribute the *excess* at v among the edges leading away from v. A generalization of the series-parallel decomposition tree, called a *factoring* [48], can be used to guide the excess toward the sink. In general, we push as much of the excess as possible through an arbitrary edge out of v and keep pushing in a depth-first manner toward the sink. There are two differences with the standard preflow-push approach. First, whenever we encounter a vertex w that is not dominated by v, we push the excess from other parts of the dag toward w before pushing any flow out of w (thus we guarantee the maximum possible excess at w before pushing flow out of w); the factoring appears to be a valuable tool for guiding this excess. Second, rather than pushing flow backwards when we find that we're unable to push the excess at w forward, we reroute the excess by backtracking to a choice point x (vertex for which a node reduction is required), which does not dominate w; again, factoring appears to help with finding the right x. This is still in the intuitive stage and may not lead to anything, but I feel that it's worth pursuing. The existing flow algorithms do not appear to take advantage of the special structure of dags.

Dag reduction may also play a role in obtaining simpler algorithms for maximum matching. The central issue here is whether we can restrict the number of ways to match the nodes that are removed by node reduction. This is a special case of a more general issue raised for graph problems by Lakshmipathy and Winklmann [31]: given a graph $G = (V, E)$, a decision problem P defined on G, and subsets V_1 and V_2 of V, such that $V_1 \cup V_2 = V$ and $|V_1 \cap V_2| = s$, how many bits, as a function of s, does a machine knowing only the subgraph induced by V_1 need to transmit to a machine knowing the subgraph induced by V_2 so that the second machine can give the correct answer for $P(G)$? For many NP-complete problems, a lower bound exponential in s can be shown. Many problems in P have upper bounds polynomial in s. The communication complexity of bipartite matching in this model is open. Resolution of the communication complexity of matching may lead to linear-time algorithms for planar matching, NC algorithms for general matching, simpler sequential algorithms for general matching, and a resolution of the red-green matching problem (in red-green matching, the edges are colored red or green by the input and the object is to find a perfect matching with a specific number of red edges; this problem was first posed by Papadimitriou and Yannakakis [37], in

connection with constrained spanning tree problems, and was shown to be in RNC by Karp et al [28]; no deterministic polynomial time algorithm for it is known). An exponential lower bound would be quite surprising, and would suggest that no "simple" algorithms for matching exist.

8. Research Transitions. Since much of this research is motivated by practical applications and I am in direct contact with people working on the practical issues, transition is often my primary research task. The most difficult transitional issue for me has not been one of communicating theoretical results to practitioners – I do that all the time, most often for theoretical results that are not my own. The problem has been one of translating transitional work into publishable papers. I become familiar enough with applications to understand the underlying theoretical issues, but not familiar enough to understand all the history, lore, and lingo of the application, i.e. to be able to publish results in a journal devoted to the application. Joint papers with applications people are a possibility, and I'm doing some of that now. It's sometimes hard for me to resist the temptation to nitpick at everything that doesn't have a solid theoretical foundation, and the people I work with are often slowed down by my participation in papers, proposals, and on PhD thesis committees. The nitpicking is important and usually leads everyone involved to a better understanding of the central issues. More journals and conferences devoted to the interface between theory and practice might help, and there's already a trend in that direction.

9. Technological Impacts. *This research would be aided by software tools for* animation of graph algorithms. As a step in that direction, I have proposed the following independent study project for a student. Design and implement a system that allows user to input directed or undirected graphs using a mouse pointing device in conjunction with X-Windows. The system should support such standard operations as adding a vertex, creating an edge between two vertices, deleting an edge or vertex, labeling an edge or vertex, and moving a vertex to a different screen position. Output should be a graph in adjacency list or adjacency matrix format, accessible to an algorithm implementation. Algorithm animation systems for graph algorithms exist (see, for example, [7]), but require tremendous programming effort to custom tailor for any specific application. My goal is something simple and modest, with few bells and whistles but lots of flexibility. In particular, I need to be able to add features as the need for them arises (rather than working around complicated features of an existing system to meet my needs).

The work on hypercube embedding has already benefited from having a reasonably powerful CPU available for exhaustive searches and simulated annealing trials. The SUN workstation purchased with funds from this project has been a major help in this regard.

10. Societal Issues. I would like to begin this section by thanking the Office of Naval Research for their support. There are three ways in which this ONR grant has made a major difference in my research career. First, it enabled me to entice one of our better graduate students to stay on for a PhD, rather than quitting with a master's

degree. I would like to emphasize that support of good students should be the primary motivation for obtaining research money. The more research money can be funneled directly to students, the better off we will all be in the long run. This suggests that it's better to support many smaller projects rather than fewer big ones. Second, the ONR grant has enabled me to purchase a Sun workstation for computational experiments, editing, and preparing papers. Though initially my productivity was almost halted while I figured out what kind of workstation would meet my needs and how to use the thing once I got it, it has been a tremendous help in the more recent past. A major problem at this university is lack of software support personnel – I waited several months for someone to install key pieces of software (e.g. LaTeX and X-Windows) on my station and finally had to do it myself. Staff, both secretarial and technical, should also be a category of high priority; without staff, equipment does not get optimum use and researchers spend too much of their time on routine tasks. Finally, the ONR grant has been a "shot in the arm" to my self-esteem as a researcher. It's sometimes hard to feel theoretical work is worth anything in a place that puts great emphasis on large practical projects with specific missions. If basic research is to survive and continue to contribute to our economic health (by providing practically applicable results and by training the next generation of scientists), funding agencies need to pay more attention to the quality and enduring nature of the research being performed rather than its immediate applicability to practical problems.

REFERENCES

[1] M. AUSLANDER, G. CHAITIN, A. CHANDRA, J. COCKE, M. HOPKINS, AND P. MARKSTEIN, *Register allocation via coloring*, IBM Technical Disclosure Bulletin, 24 (1981), pp. 336 – 346.

[2] S. BAASE, *Computer Algorithms: Introduction to Design and Analysis, Second Edition*, Addison-Wesley, 1988.

[3] C. BERGE, *Graphs and Hypergraphs*, North Holland, 1976.

[4] M. BERN, E. LAWLER, AND A. WONG, *Linear-time computation of optimal subgraphs of decomposable graphs*, Journal of Algorithms, 8 (1987), pp. 216 – 235.

[5] S. BHATT, F. CHUNG, F. LEIGHTON, AND A. ROSENBERG, *Efficient embeddings of trees in hypercubes*, Typescript, Department of Computer Science, Yale University, New Haven, CT 06520.

[6] S. BOKHARI, *On the mapping problem*, IEEE Transactions on Computers, C-30 (1981), pp. 207 – 214.

[7] M. H. BROWN, *Algorithm Animation*, MIT Press, 1988.

[8] G. CHATIN, *Register allocation and spilling via graph coloring*, in Proceedings of the SIGPLAN Symposium on Compiler Construction, 1982, pp. 23 – 25.

[9] W. CHEN, *Experiments with heuristics for hypercube embedding*, Project report for CSE 691I: Surviving Intractability, Spring, 1989, North Carolina State University.

[10] W. CHEN, *A Graph-Oriented Mapping Strategy for a Hypercube*, Master's thesis, North Carolina State University, 1988.

[11] W. CHEN, E. GEHRINGER, AND M. STALLMANN, *Hypercube embedding heuristics: an evaluation*, In preparation.

[12] W. CHEN AND M. STALLMANN, *Analysis of a heuristic for embedding binary trees into hypercubes*, In preparation.

[13] F. CHOW AND J. HENNESSY, *Register allocation by priority-based coloring*, in Proceedings of the SIGPLAN Symposium on Compiler Construction, 1984, pp. 17 – 22.

[14] E. DAHLHAUS AND M. KARPINSKI, *The Matching Problem for Strongly Chordal Graphs is in NC*, Tech. Rep. 855-CS, University of Bonn, 1986.

[15] S. ELMAGHRABY, J. KAMBUROWSKI, AND M. STALLMANN, *On the reduction of acyclic digraphs and its applications*, Working paper.

[16] F. ERCAL, J. RAMANUJAM, AND P. SADAYAPPAN, *Task allocation onto a hypercube by recursive mincut bipartitioning*, Typescript, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio 43210.

[17] S. EVEN AND R. TARJAN, *Computing an st-numbering*, Theoretical Computer Science, 2 (1976), pp. 339 - 344.

[18] M. FELLOWS AND M. LANGSTON, *Nonconstructive advances in polynomial-time complexity*, Information Processing Letters, 26 (1987), pp. 157 - 162.

[19] S. FORTUNE, J. HOPCROFT, AND J. WYLLIE, *The directed subgraph homeomorphism problem*, Theorectical Computer Science, 10 (1980), pp. 111-121.

[20] A. GOLDBERG AND R. TARJAN, *A new approach to the maximum flow problem*, Journal of the ACM, (1988).

[21] L. GOLDSCHLAGER, R. SHAW, AND J. STAPLES, *The maximum flow problem is log space complete for P*, Theoretical Computer Science, 21 (1982), pp. 105 - 111.

[22] X. HE AND Y. YESHA, *Binary tree algebraic computation and parallel algorithms for simple graphs*, Journal of Algorithms, 9 (1988), pp. 92 - 113.

[23] C. HSU, *Minimum via topological routing*, IEEE Transactions on Computer Aided Design, CAD-2 (1983), pp. 235 - 246.

[24] S. HUANG, *A fully pipelined minimum cost spanning tree constructor*, Journal on Parallel and Distributed Computing, (1989). To appear.

[25] D. S. JOHNSON, *The NP-completeness column: an ongoing guide*, Journal of Algorithms, 6 (1985), pp. 434 - 451.

[26] D. S. JOHNSON, C. R. ARAGON, L. A. MCGEOGH, AND C. SCHEVON, *Optimization by simulated annealing: an experimental evaluation (part I)*, Typescript.

[27] J. KAMBUROWSKI AND M. STALLMANN, *Reducing transitive vertex cover to bipartite vertex cover*, Submitted to Information Processing Letters.

[28] R. KARP, E. UPFAL, AND A. WIGDERSON, *Constructing a perfect matching is in random NC*, in Proceedings 17th Annual ACM Symposium on Theory of Computing, 1985, pp. 22 - 32.

[29] S. R. KOSARAJU, *Speed of recognition of context-free languages by array automata*, SIAM Journal on Computing, 4 (1975), pp. 331 - 340.

[30] M. KRISHNAMOORTHY AND N. DEO, *Complexity of the minimum-dummy-activities problem in a PERT network*, Networks, 9 (1979), pp. 189 - 194.

[31] N. LAKSHMIPATHY AND K. WINKLEMANN, *"Global" graph problems tend to be intractable*, Journal of Computer and System Sciences, 32 (1986), pp. 407 - 428.

[32] S. LEE AND J. AGGARWAL, *A mapping strategy for parallel processing*, IEEE Transactions on Computers, C-36 (1987), pp. 433 - 442.

[33] A. LEMPEL, S. EVEN, AND I. CEDERBAUM, *An algorithm for planarity testing of graphs*, in Theory of Graphs: International Symposium, July, 1966, P. Rosenstiehl, ed., Gordon and Breach, New York, 1967, pp. 215 - 232.

[34] J. LEWIS, *On the complexity of the maximum subgraph problem*, in Proceedings 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 265 - 274.

[35] G. MILLER AND J. REIF, *Parallel tree contraction and its application*, in Proceedings 26th Annual Symposium on Foundations of Computer Science, 1985, pp. 478 - 489.

[36] B. MONIEN AND I. SUDBOROUGH, *Simulating binary trees on hypercubes*, in VLSI Algorithms and Architectures: 3rd Aegean Workshop on Computing, 1988, pp. 170 - 180.

[37] C. PAPADIMITRIOU AND M. YANNAKAKIS, *The complexity of restricted spanning tree problems*, Journal of the ACM, (1982), pp. 285 - 309.

[38] V. RAMACHANDRAN, *The complexity of minimum cut and maximum flow problems in an acyclic network*, Networks, 17 (1987), pp. 387 - 392.

[39] C. RIM, T. KASHIWABARA, AND K. NAKAJIMA, *A note on the NP-hardness of the topological via minimization problem*, 1989, Typescript.

[40] N. ROBERTSON AND P. SEYMOUR, *Disjoint paths - a survey*, SIAM Journal on Algebraic and Discrete Methods, 6 (1985), pp. 300 - 305.

[41] ——, *Graph minors - a survey*, in Surveys in Combinatorics, I. Anderson, ed., Cambridge University Press, 1985, pp. 153 - 171.

[42] M. SARRAFZADEH AND D. LEE, *A new approach to topological via minimization*, 1989, To appear, IEEE Transaction on Computer-Aided Design.

[43] C. SAVAGE, *A systolic design for connectivity problems*, IEEE Transactions on Computers, C-33 (1984), pp. 99 - 104.

[44] C. SAVAGE, M. STALLMANN, AND J. PERRY, *Solving some combinatorial problems on arrays with one-way dataflow*, Algorithmica, (1989). To appear.

[45] M. STALLMANN, *Maximum node coloring of graphs with special structure*, 1989, In preparation.

[46] ——, *PQ-trees, book embeddings, and planar embeddings*, May 1989, Extended abstract.

[47] M. STALLMANN AND W. BEIN, *Decomposition and reduction of directed acyclic graphs (revised version)*, In preparation.

[48] ——, *Decomposition and reduction of directed acyclic graphs*, July 1988, submitted to SIAM Journal on Computing.

[49] M. STALLMANN, T. HUGHES, AND W. LIU, *Unconstrained Via Minimization for Topological Multilayer Routing*, Tech. Rep. C88495, Semiconductor Research Corporation, P.O. Box 12053, Research Triangle Park, NC 27709, 1988. Revised version submitted to IEEE Transactions on CAD, June, 1989.

[50] K. SUPOWIT, *Decomposing a set of points into chains, with applications to permutation and circle graphs*, Information Processing Letters, 21 (1985), pp. 249 - 252.

[51] M. TCHUENTE AND L. MELKEMY, *Reseaux Systoliques pour le Calcul des Composantes Connexes et le Triangularisation des Matrices Bandes*, Tech. Rep. 366, Laboratoire d'Informatique et de Mathematiques Appliquees de Grenoble, 1983.

[52] W. UNGER, *On the k-colouring of circle graphs*, in Proceedings 5th Annual Symposium on Theoretical Aspects of Computer Science, 1988, pp. 61 - 72.

[53] J. VALDES, *Parsing Flowcharts and Series-Parallel Graphs*, PhD thesis, Stanford University, 1978.

[54] A. WAGNER, *Embedding Trees in the Hypercube*, Tech. Rep. 204/87, Department of Computer Science, University of Toronto, 1987.

[55] A. WU, *Embedding of tree networks into hypercubes*, Journal on Parallel and Distributed Computing, 2 (1985), pp. 238 - 249.

[56] M. YANNAKAKIS, *Node- and edge-deletion NP-complete problems*, in Proceedings 10th Annual ACM Symposium on Theory of Computing, 1978, pp. 253 - 264.

Adaptive DBMS Support for Complex Applications

Roger King

University of Colorado
Department of Computer Science
Boulder, Colorado 80309

Abstract

The goal of the adaptive database project at the University of Colorado is to develop techniques which will make database systems useful for newer applications, such as engineering design. In such cases, there is a need to support complex, computed data, as well a need to hand-tailor a database system to suit specific processing requirements, for example, version support and document management. Two different experimental systems, one addressing each of these concerns, are under construction. The algorithms and techniques developed for these systems are intended to help relieve the advanced database user from the highly constrained mechanisms which traditional database management systems provide.

1. Background

Traditionally, database systems were used by business programmers, and their needs were at least perceived to be rather simple. Data in the real world spans a wide spectrum of complexity, from highly unstructured (like text) to highly structured (like airplane designs). In data processing environments, data is typically represented only in a narrow band of this spectrum. All data is seen as being tightly, yet simply, structured. Further, most transactions against the database are submitted in batch mode. The goal is merely to support the fast retrieval of large numbers of similar, simply-structured records. As a result, conventional database systems provide very little in the way of abstraction, and in particular cannot effectively represent data whose internal structure is either highly structured or highly unstructured.

In recent years, a new generation of potential database users has emerged. This includes software engineers, VLSI and printed circuit board designers, aircraft and CAD engineers, as well as those involved in office automation. These individuals wish to store and manipulate many forms of data, in particularly, highly structured objects. (There is a need to represent unstructured data, specifically text, as well, but this research project does not address this issue.) Further, engineers often wish to manipulate data in an interactive environment. In sum, newer database users have a need for all the amenities a database system provides - such as concurrency, serializability, transaction management, rollback and recovery - but in an interactive design mode. Since traditional database systems do not suit these needs, many researchers are examining the numerous problems related to this grand challenge.

2. Research Objectives and Issues

Clearly, the goal of providing database support for interactive design users is gigantic. New data models, storage and access mechanisms, query languages, user interfaces, and many other tools are needed. In this project, we focus on two specific problems and use a common philosophical approach in attacking each of them. Our first area of concentration involves the support of computed data. In a design system, as opposed to a data processing system, there is a vast amount of tightly interconnected computed data. A design for an airplane includes highly interrelated data; changing one part of the design is likely to have effects on many other aspects. Further, it must be accessed quickly, as designers work in real time. Our second focus is on a broader issue, that of allowing advanced users to cleanly integrate into one database environment a variety of complex tools. For example, in a software development system used by software engineers, the

database must interact with versioning, configuration, and report systems.

We are approaching each of these tasks from the perspective of adaptability. This means that, unlike existing database systems, the DBMS is not rigid. In our first research effort, we are focusing on the ability of the system to adapt itself at the physical level; computed data is managed in a way that allows the DBMS to learn from past usage experience and rearrange the way it processes updates. This is crucial in minimizing the potentially exponential costs of calculating computed data. In the second effort, we focus on the ability of the database user to adapt the system to suit his or her needs - at the conceptual level. This is important, as engineering applications vary dramatically in their requirements, and often require very specialized tools.

3. Approaches and Progress

The two projects described above are called Cactis and A La Carte. Cactis has resulted in the development of parallel algorithms for the maintenance of computed (or derived) data. These algorithms are based on attributed graphs and dramatically reduce the amount of I/O necessary to keep complex engineering database entities up to date. A La Carte uses the approach of abstracting the database management system up another level, resulting in the design of a database generator; such a system is, as a result, designed to be much more tailorable. The main problem lies in doing this in a fashion which does not require vast amounts of low-level programming.

Both of these projects also share another common philosophic approach, besides one of adaptability. They both attempt to integrate two directions which have been prominent in the database research community - behavioral and structural (or "semantic")

object-oriented modeling. (Behavioral object-oriented modeling is often simply referred to by the term objected-oriented.) This has allowed the support of data objects which are both structurally complex and dynamic. This is crucial in supporting emerging engineering applications. Below, we discuss both projects, first Cactis, then A La Carte.

3.1. Cactis

Consider an engineering design application familiar to all of us: software development and reuse. In every phase of the software life-cycle, we see a need for derived data. Examples include the following data relationships: the dependency between a source module and the corresponding object module; the derivation of a load module from a number of object modules; and, the relationships between a set of software modules and the associated documentation, requirements, bug reports, fix reports, and project milestones. In each case, if one piece of data changes, others are likely to be changed as a direct consequence.

With traditional database systems, this sort of derived data must be maintained by the application software or directly by end users - typically with a mechanism known as triggers. This introduces problems. Programmers are not likely write code that is portable from one software environment to another. Also, if computed data is maintained directly by the DBMS, then it may be managed in a much more efficient and correct fashion. Cactis [7, 9] is designed to support computed data in a highly efficient manner, and to do so in a consistent fashion. Triggers, on the other hand, must be hand-coded by the user and are difficult to reuse. Even more significantly, as a trigger mechanism is likely to operate in a first come, first severed basis, no attempt is made to optimize their execution. In general, if several trigger sequences all lead to the same piece of computed

data, it could be updated an exponential amount of time, with respect to the number of trigger paths to the data item. A prototype Cactis system has been implemented, in order to provide a basis for the experimentation with and evolution of the underlying algorithms. In particular, substantial experiments have been performed in order to illustrate that the techniques developed are useful for engineering databases. The research is being conducted in conjunction with Scott Hudson of the University of Arizona.

Cactis represents a database as an attributed graph, and uses an incremental graph update algorithm. It also is self-adaptive, in that it learns from past experience and adjusts both process scheduling and data clustering on disk to minimize the I/O cost of maintaining computed data. We have run extensive performance tests on Cactis, illustrating substantial savings when the system is used. The potentially exponential behavior of triggers has been reduced to linear cost.

Also, several components of a software environment, including a "Make" [5] facility, a critical path tool, and a bug report system have been built on top of Cactis. Further, the Arcadia software environment project [6, 10, 12] has made some use of Cactis.

Cacti [8] is a distributed version of Cactis, and is currently under construction. It is targeted for a local network of Sun workstations, and is motivated by the fact that software design teams often work in distributed, interactive environments. The implementation of the system is being greatly facilitated by the fact that the graph algorithm in Cactis is naturally parallel, thus making it easy to adapt it to a distributed environment. In keeping with the self-adaptive nature of Cactis, the new system uses usage statistics to replicate, migrate, and recluster data around the network.

3.2. A La Carte

A La Carte [2] is in its early stages, and addresses much higher-level issues than Cactis or Cacti. The project, which is being conducted in conjunction with Colorado PhD students Pam Drew and Jonathan Bein, was motivated by the lesson that Cactis is still a very low level tool, and that many problems arise when trying to integrate various software environment tools within a Cactis application. Again, a prototype is under development, so that real experiments can be performed to validate and evolve the techniques under design.

The system uses mixins and multiple inheritance to allow an engineer to select both database facilities and software environment capabilities. For example, the designer of a software environment may choose an appropriate concurrency control option and clustering mechanism, as well as a version facility, a document management mechanism, and a configuration tool. A La Carte puts them all together in one system, using a method integration technique. It thus is very similar in spirit to Exodus [3] and Genesis [1]; a significant difference is that A La Carte is a less aggressive project, and is oriented mostly toward examining the appropriate mechanisms for resolving conflicts when mixing in complex software methods.

4. Research Directions

There are many, many other aspects of database support for newer complex data that must be investigated. Of prime importance is the representation, in a coordinated fashion, of different levels of structured data, all the way from text to video to sound to graphical images to layout diagrams. A few researchers are working on multi-media

database projects oriented toward solving this problem [4]. In particular, many engineering applications have very demanding data modeling requirements. Consider PCB boards; the job of representing the wiring problem is immense, and current wiring software represents the board as an unstructured file, with all the semantics of the board embedded in an ad hoc fashion in the application software. We hope that Cactis and Cacti will provide some help in maintaining the relationships between various forms of data. As another example, if a CAD image changes, the documentation describing it is likely to change as well. A La Carte should also provide some insight, in terms of providing a mechanism for integrating the wide variety of tools needed to support multiple forms of data.

Engineers also want to manipulate computed data in real-time, using interactive, staged transactions. Traditionally, DBMS's have been tailored toward the support of batched transactions. And, in the future, when a design error occurs, it will not be sufficient to blindly rollback the entire transaction (which may have taken days or weeks). For example, if a piece of source code is changed, this might automatically affect documentation, milestones, bug reports, test data, and test executions. If one of these executions abnormally terminates, the software designer does not want the entire transaction - including the source code changes - backed out. Very fine-grained control of how the transaction is indeed backed out is needed. Some aspects of staged recovery can be viewed as layered forms of derived data; Cactis and Cacti might be of help in dealing with this this. A La Carte might be useful in assisting the database user in choosing and integrating specific forms of recovery.

The design engineer will also want less restrictive forms of concurrency control. If a data item is currently in use, and another engineer wishes to use it, he would rather electronically tap the current user on the shoulder and ask for a copy of the item. Current concurrency control algorithms will make him wait an arbitrary amount of time. This is due to the assumption that database transactions are done in batch mode. A related issue concerns versions (a form of derived data, a sort of which is supported by Cactis and Cacti). Will versioning replace concurrency control in interactive systems? Perhaps rather than locking an item, a user will merely version it. A user might check out a copy and check it back in to create a new "current" version, in much the same way as engineers now check in and check out design documents. This of course brings up the age-old problem of version integration. This is related to the problem of reversing the process of creating derived data. If you change a piece of derived data (a version) how does it affect the original?

Also, as structural encapsulation (semantic modeling) and behavioral encapsulation (object-oriented modeling) become more popular, a challenging question will arise. How will the many forms of behavioral encapsulation be integrated? Or better yet, should they? Cactis and Cacti use derived data as their behavioral techniques. A La Carte uses methods. They are very similar, but not identical. There is no message passing paradigm in Cactis and Cacti, and a method does not normally store the result of its actions, unless the user codes it that way. The problem does not stop there. How do methods and derived data relate to rule systems and constraint languages? Will we want a DBMS to support all four of these techniques or just one? And if we want to mix them, what theoretical and algorithmic work needs to be done? A related issue is that of integrating more complex forms of manipulations (such as methods, rules, constraints, and derived

data) with traditional set-oriented and aggregate operations; after all we do not want to lose the capability that relational systems are so good at.

One interesting question is how all this new technology could be used by business environments. I believe that data processing systems do indeed represent complex objects, versions, methods, rules, and derived data - they are just accustomed to embedding them in application software and have never had more powerful tools available.

Probably one avenue that (in my opinion) should not be pursued is the perverting of current relational paradigms to solve these problems. The various attempts by some researchers to make relational systems look like semantic systems were very unsuccessful. This is due to the inherent lack of abstraction in the relational model. The key problem is that the relational model has no concept of an object - all database items are made up of identifiers and cannot be recursively combined into complex objects. I believe that current attempts at taking the relational model and making it look (behaviorally) object-oriented will fail for the same reason. No matter what, the user will be conscious of manipulating identifiers - not objects.

5. A Grand Challenge

A few months ago, a group of about forty researchers met for two days in the Napa valley [13]. All participants were active researchers in either the software engineering or database realm. There were no scheduled presentations. The goal was to determine, as a working group, the research tasks which needed to be performed in order to provide effective database support for software environments. (A software environment was generally accepted as a software system which assists program developers in the design, cod-

ing, debugging, deployment, maintenance, and eventual reuse of software systems.) Many different ideas were discussed - but a surprising result came out of the workshop. It was generally felt that the biggest challenge was the *integration* of all of the many research results that are currently being published in the software environment database area.

Indeed, the central problem seems to be that there is no underlying, integrated model or representation for software database support. Many authors feel that "object-oriented" databases are the answer - but no one could agree on the definition of the term. And, unlike other, more mathematically tractable models such as the relational model, there is no clear way of representing the implementation of object-oriented databases. The same is true for object-oriented query specification and optimization. My feeling is that object-oriented databases are *not* at all *identical* to engineering databases; it is merely true that the object-oriented paradigm is a promising platform for studying engineering issues.

Furthermore, while many researchers are working on new ways of managing large and complex objects, of executing long and interactive and staged transactions, of interfacing graphically with a complex database, and of implementing novel forms of concurrency control - it is not clear what challenges lie in putting all these things in one system. This would require a very large scale research platform. And, it is clear that these various new software mechanisms will interact in as yet unknown ways. The big question is: Will a uniform, understandable, and implementable theory of object-oriented database design and implementation arise? Indeed, our hope is that Cactis, Cacti, and A La Carte are a step in this direction.

6. Research Transitions

My own goal is to link up as much as possible with existing engineering projects, to see if the results of my work are usable. The DARPA sponsored project Arcadia [11] is currently my main target project. Already, I have learned a few things. For example, the algorithms used in Cactis to schedule derived computations are viewed as too restrictive by many engineers. They would like more control over how the system makes decisions. This can benefit the system by providing crucial information that may be very hard to deduce automatically. An example is that a user may know that he is about to suddenly shift his area of focus, and if he is able to warn the database, Cactis can adapt more quickly.

7. Technological Impacts

The biggest technological area that will impact this work is parallel and distributed computing. The ready availability of high-speed networks and workstations, the growth of long-haul networks, the development of many-processor machines, the introduction of parallel channels, and the development of disk arrays will all affect database technology. This is why the algorithms in Cactis were designed to be naturally parallel.

8. Societal Issues

It goes without saying that ONR support has dramatically affected my career. Early and intensive financial support enabled me to get my research program off the ground and quickly produce solid results. Further, two of my PhD students are now professors (Scott Hudson of the University of Arizona and Nabil Kamel of Michigan State), and I

am currently working with ten other PhD students. Of course, I am sure that not all of them will stay with me, but students are naturally attracted to large, active research programs. In sum, I am a very strong supporter of the YIP program.

The only other societal issue I might point out is that the grand challenge mentioned above will necessitate a much more coordinated and cooperative effort by American researchers. If we were able to set research goals at a national level (as the Japanese do), and corporations would not always operate in a strictly product-based manner, much more progress could be made.

9. Recommendations to Funding Agencies

My main recommendation is that federal funding agencies should not try to set specific research goals and support only projects related to these goals, unless they are also prepared to help researchers coordinate their efforts. Doing the first without doing the second only produces many fragmented projects which do not build on each other. I think that funding larger, multi-institution projects is a good idea.

References

1. D. S. Batory, J. R. Barnett, F. F. Garza, K. P. Smith, K. Tsukauda, B. C. Twichell and T. E. Wise, "GENESIS: A Reconfigurable Database Management System", *To appear in IEEE Transactions on Software Engineering*, .
2. J. Bein, P. Drew and R. King, "A La Carte: A Generator of Persistent Object Stores for Software Environments", *Working paper*, 1988.
3. M. J. Carey, D. J. DeWitt, D. Frank, G. Graefe, M. Muralikrishna, J. E. Richardson and E. J. Shekita, "The Architecture of the EXODUS Extensible DBMS",

Proceedings of the Workshop on Object-Oriented Databases, Pacific Grove, California, September 23-26, 1986, 52-65.

4. S. Christodoulakis, "Multimedia Database Management Systems (Panel Statement)", *Proc. ACM SIGMOD Int. Conf. on the Management of Data*, 1985, 304-305..

5. S. I. Feldman, "Make -- A Program for Maintaining Computer Programs", *Software - Practice and Experience* 9 (April 1979), 255-265.

6. D. Heimbigner and S. M. Sutton, "SCAD Support for Software-Process Programming", *ACM Software CAD Databases Workshop*, Napa, California, February 27-28, 1989.

7. S. Hudson and R. King, "The Cactis Project: Database Support for Software Engineering", *IEEE Trans. on Software Engineering*, June 1988.

8. S. Hudson and R. King, "An Adaptive Derived Data Manager for Distributed Databases", *Second International Workshop on Object-Oriented Databases*, Bad Munster am Stein-Ebernburg, FRG, September 1988, 193-203.

9. S. Hudson and R. King, "Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System", *ACM Transactions on Database Systems*, 1989.

10. S. M. Sutton, D. Heimbigner and L. J. Osterweil, "Programmable Relations for Managing Change During Software Development", *University of Colorado at Boulder Technical Report CU-CS-418-88*, September 15, 1988.

11. R. Taylor, "Arcadia: A Software Development Environment Research Project", *University of California at Irvine, Dept. of Information and Computer Scinece, Technical Report*, April 1986.

12. R. N. Taylor, F. C. Belz, L. Clarke, L. J. Osterweil, R. W. Selby, J. C. Wileden, A. Wolf and M. Young, "The Arcadia Environment Architecture", Tech. Report, Univ. of Calif.at Irvine, Dept.Info.and Comp. Sci, Irvine, Calif., Sept. 1987.

13. *ACM SIGMOD Software Eng. Notes Software CAD Databases Workshop*, Napa, California, February 27-28, 1989.

TRANSFORMATIONAL PROGRAMMING SYSTEMS WITH LARGE-SCALE AUTOMATION

Robert Paige

0 ,1) Abstract and Background

We regard the discovery of useful program transformations as part of a natural evolutionary process that begins with the discovery of informal principles of software engineering and leads to their formalization and mechanization within a compiler. For this process to succeed we believe that three essential methodologies must be developed - for programming, transformations, and compilers. Programming methodology is the informal but essential principles that facilitate the manual construction of programs and the synthesis of algorithms. Transformational methodology is the more formal process of syntactic analysis and symbolic manipulation by which programs can be improved automatically or semiautomatically. Compiler methodology is a fully automatic form of transformational methodology used to implement a programming language.

There is a natural process whereby programming methodology matures and devolves into transformational methodology whose perfection leads to compiler methodology. This process is behind the evolution of low level machine languages into high level languages. This process involves the recognition that major common patterns of programming style in a low level language can result from the application of standard techniques of program improvement to higher level programming prototypes. When such a technique can be formalized as an implemented 'meaning-preserving' program transformation, then we can conveniently write programs at a higher level of abstraction without a penalty in performance, since the efficient lower level versions can be derived mechanically or semiautomatically by transformation.

Such transformations form the essential lemmas in a proof of correctness of the implementation-level code. When these transformations can also be associated with a precise measure of improvement in time and space, then the lemmas can facilitate a proof of performance as well as functionality of the low level code. This approach to verification leads naturally to the intriguing idea of problem specification languages that fall within specific complexity classes; i.e., that can be compiled automatically into implementations whose time and space bounds are guaranteed at language design time. In this approach the assumed correctness of a specification together with a correctness proof of the 'supercompiler' proves the functionality and the performance of the compiled code.

2) High Level Research Objectives

a. to automate major aspects of programming

b. to develop a new means of software production that enables us to implement algorithms or construct software that would be too complex to program with existing technology

c. to unify problem specification, program design, verification, and analysis within a single unified framework

d. to make it easier to teach and understand algorithms and software engineering

3) Research Issues

(see Appendix I)

4) Technical Approach

a. The recent focus of our project is in developing a new paradigm of 'program verification by compilation'. Within this paradigm programs are verified for their functionality and performance. This is achieved by defining a problem specification language L and a compiler for L that will always generate code with guaranteed time and space complexity. Thus, a single theorem stating the correctness of the compiler for L is sufficient to guarantee the functionality and performance of any program compiled from an L specification.

b. Program transformations such as fixed point iteration, finite differencing, stream processing, and real-time set machine simulation on a RAM have been developed within our project and comprise the major phases of the compiler mentioned in part (a) above.

c. Underlying any programming methodology is the theory of algorithms. Algorithm implementations are also the most difficult kinds of programs. Thus, we are benefitting greatly by studying algorithms and their derivation by transformation in order to develop improved programming and transformational methodologies.

d) We are making use of real-time simulation of an abstract set machine on a RAM for data structure selection. This approach seems to be new and promising.

e) Rather than use attribute grammars for semantic analysis, we have chosen a pattern directed approach similar to MENTOR/TYPOL.

5) Progress

a. With Cai we have developed a functional problem specification language called SQ+ (SETL expressions plus fixed points) that can express all partially recursive functions. We show how to compute fixed points efficiently for abstract SQ+ functions over abstract lattice theoretic data types.

b. In POPL87 Cai and I reported on a subset of SQ+ called

L1 that can always be compiled into programs that run on a RAM in asymptotically linear time and space with respect to the problem input/output space. We showed that a significant fragment of an optimizing compiler could be specified in L1. Recently, Cai has shown that the very difficult problem of Planarity Testing can also be specified in L1.

c. In our algorithm research with Tarjan we have have focussed on partition refinement as an algorithmic strategy and have discovered improved algorithms for the Single Function Coarsest Partition Problem, Lexicographic Sorting, the Relational Coarsest Partition Problem, and Double Lexical Ordering.

d. Cai and I have generalized Hoffman and O'Donnell's fast top-down tree pattern matching algorithm to be incremental with respect to patterns and to handle more general patterns with several pattern variables and implicit equality testing. Cai used this algorithm to design and implement an inductive definition language to be used for semantic analysis in RAPTS. Tarjan and I recently obtained some space time tradeoffs for a potentially useful subclass of these patterns.

e) We recently developed a methodology for implementing sets and maps on a conventional RAM based on real-time simulation. We intend to use this work as the basis for the third phase of the L1 compiler, which should be operational by the end of the summer.

6) Research Directions

a) Can our transformational methodology be applied to machine models other than the sequential RAM? What about for parallel RAM's?

b) Can we see a great improvement in reliability, performance, and labor costs in constructing software using our methodology. In particular can we construct the major components of a high performance optimizing compiler in a relatively brief time period by transforming a mathematical specification.

c) Can our transformations be usefully generalized and implemented with faster and better algorithms? Tree pattern matching seems to be a fundamental operation within our methodology Can our matching algorithms be improved further? Are there better heuristic or approximation algorithms to solve those problems that are NP-hard: e.g., stream processing?

d) Can our techniques be further applied to database optimization and integrity control?

e) Can we generalize our results with L1 to develop a whole heirarchy of problem specification languages for complexities at each polynomial degree? Is there a useful subset of Prolog equivalent to L1? Can we define a syntactic class of attribute grammars (even circular

grammars) that fall within L1?

7) One grand practical challenge would be to be able to implement a high performance (comparable to compile- and run- times for IBM's best compiler) optimizing FORTRAN compiler for an IBM 370 by generating it from a succinct in L1 specification. An ambitious theoretical challenge would be to produce specification languages with worst case complexity bound to each polynomial degree and whose union has the same power as Gurevitch and Shelah's polytimeIO language.

8) i) Our work has had an impact mostly on the transformational programming community, but also on the database, programming language, algorithm, and complexity communities.

We had a major impact on a European ESPRIT project in rapid prototyping called SED with participants from Thomson-CSF and INRIA in Paris, Enidata in Rome, U. of Patras in Greece, Hildesheim U. in W. Germany.

IFIPS WG2.1, the original ALGOL working group, is currently developing a new mathematical problem specification and programming language within a transformational programming environment. The group has been influenced by our work and has asked me to head a subgroup on program transformations including finite differencing.

The Dutch government has begun a national project on transformational programming. I was invited to the Netherlands to present the results of our project. Representatives from the Prospectra Project, project CIP, Kestrel, ISI, and other groups seem to have been influenced by our work in finite differencing and want to utilize our current work in fixed point iteration and data structure selection by real-time simulation.

A group of researchers including Neil Jones from DIKU working with the transformational paradigm of mixed computation feel some impact of our work and, consequently, I am an invited speaker at the ESOP conference in Copenhagen in May 1990.

The database community (e.g., N. Roussopoulos) seems to have been influenced by our earlier work in integrity control.

The language theory community seems to be interested in the algorithm work with Tarjan on the relational coarsest partition problem

We inspired Gurevich and Shelah to investigate function classes computable with respect to input and output space.

ii) Before anyone else uses the results of our technology, it would be most convincing if we would be able to use it ourselves. This is only now starting to happen. By the end of the summer when we expect our L1 compiler to generate

linear time C code, we will be in a better position to assess how the technology can best be applied and by whom. Our hope is that we can consider implementing code for the forthcoming I80860 Intel chip, a 1 million gate RISC mini-Cray I.

9) The RAPTS project would benefit from a few SUN 3/60's and half of a super eagle disk. SUN 3/60's could overcome some of the speed problems we have with SETL as our main implementation language. It would also make it easier to run the prototyping systems MENTOR and the SED environment, which are relevant to our work. However, space is a more serious problem for us, because of SETL and also our reliance on large tables to facilitate fast pattern matching to implement transformations.

10) This year there too many qualified faculty applicants for too few positions in the U.S. At NYU, where we received over 200 applications, we have been reluctant to send out rejection letters. If this situation continues, I believe that it would be humane and also scientifically sound for funding agencies to allocate more funding for post-doc positions and less for students.

11) Recommendations on Funding

Transformational programming and parallel computation are two emerging fields that may ultimately depend on each other for success. Perhaps, because ad hoc programming on sequential machines is so straightforward, sequential programming methodology has had little impact outside the academic community, and transformational methodology has had little impact at all. However, because ad hoc programming for parallel machines is so hard, and because progress in software construction has lagged behind architectural advances for such machines, there is a much greater need to develop parallel programming and transformational methodologies. ONR should stimulate research on formal ways to overcome problems of parallel computation -with respect to both software development and algorithm design.

Specific problems include derivations of synchronous and asynchronous parallel algorithms, systems, and architectures. These derivations may include sequential to parallel machine translation, retiming and reconfiguration techniques, and techniques for simulation of one kind of parallel machine in another. Incorporation of specification, design, verification, and analysis into parallel programming, transformational, and compiler methodologies. Also applications of transformational and derivation techniques: for explication of difficult parallel algorithms(e.g., parallel graph algorithms), and to solve real-world problems are of special interest (e.g., derivations of communication protocols and database query optimization for distributed computation).

Appendix I. Research Issues

Hypothesis: The theory of algorithms underlies the science

of programming. A viable theory of problem specification and program transformationn would provide the basis for a theory of algorithm and program design.

Main Objective: The discovery of basic program transformations that capture principles of algorithm design; implementation and application of these transformations within the RAPTS system.

Motiviation for Research:

Four general difficulties with current program construction methodologies (originating with Knuth[68] and Aho, Hopcorft, and Ullman[74]) together with long range goals of our project are listed below.

1. (Problem Specification)

Problems are specified in an ad hoc way. But informal problem specifications can be confusing, even ambiguous. This is bad for documentation and complicates the synthesis and analysis of correct programs.

Our goal is to provide a formal mathematically based problem specification language.

2. (Program Synthesis)

Programs are constructed informally.

Our goal is to directly map problem specifications into efficient implementations by applying correctness preserving transformations.

3. (Program Correctness)

Program proofs are tied closely to implementation level code. Consequently, they are lengthy, complicated, and unconvincing.

Our goal is to guarantee the correctness of an implementation by proving the correctness of the problem specification and the transformations used to derive the implementation.

4. (Program Analysis)

Time and space complexity of implementations depends on a rigorous analysis of the low level features of the implementation code, undependent of any design principles.

Our goal is to integrate performance analysis with the synthesis process.

Program construction using a supercompiler:

Abstract Problem Specification (supplied by user-

Main Sources of transformational programming methodology:

Topdown Stepwise Refinement: Dijkstra and Wirth[late 60's]
General Idea for a system: Cheatham[72]
Correctness: Floyd[67], Hoare[69]
Transformational Correctness: Gerhart[75]
Mechanical Performance Analysis: Ramshaw[79],J. Cohen[82]
Performance Analysis by Transformation: Willard[83]
Specification Language:
VERS2-Earley[74], SETL-Schwartz[77], LCF/ML-Gordon,Milner, Wadsworth[79], Algorithmic Language-Bauer[82]
Transformations:
Recursion to Iteration-Walker,Strong[73]
Dynamic Programming-Bird[80],N. Cohen[83]
Stream Processing-Allen,Cocke[71],Morgenstern[76],Burstall and Darlington[77],Reif,Scherlis[82]
Finite Differencing-Briggs[16th century],Cocke,Schwartz[69], Cocke,Kennedy[77],Earley[76],Fong,Ullman[76],Fong[77,79]

Related work in formal program development methodology

1. ad hoc program construction and formal verification and proof checking

   Manna
   Luckham
   etc.
   see Lipton, demillo, perlis for criticism

2. synthesis
   constructive proof
   Manna and Waldinger
   Goad
   Bibel
   Bates

   Much manual intervention - efficiency is not considered

   Equational Approach
   Guttag
   Huet
   Hoffman and O'donnell

Efficiency is of even less concern - but mechanization is a plus

3. Transformational

Bauer
Cheatham
Burstall and Darlington

Much manual intervention, vast transformational libraries, long aimless derivation sequences, unpredictable capacity for improvement

Case Study

Three basic program transformations of wide applicability have been developed and implemented within RAPTS. Development of a fourth basic transformation is in progress. They are illustrated below.

1. Solving Roots of Set Theoretic Predicates -- The Genesis of Algorithmic\Strategy

Restrictions

Determinate problems
Executable Specs
Finite Sets

determinate problems such as these are reminiscent of Linear Programming, but the solution method we devise is similar to the iterative techniques used to find approximate solutions to numerical equations.

find the unique solution S by the following iterative procedure:

The solution method above is based on Tarski. When it can be applied, the solution to the original problem is guaranteed to run in polynomial time.

Our transformation is described in Paige[84,84]. Generalization of this transformation to finding solutions in partition spaces played a central role in the explanation of a new improved algorithm to solve the single function coarsest partition problem Paige,Tarjan[84].

2. Finite Differencing -- The Efficient Implementation of Strategy

Further improvement to the attribute closure procedure generated by the previous transformation can be achieved by automatic application of finite differencing transformations, so-called because they derive from Briggs's 16th century method of polynomial tabulation using difference polynomials. But instead of tabulating polynomials, we want to tabulate expressions of various data types. For this example, instead of computing the expression ***** in the naive way each time through the loop, we will tabulate the value of this expression for each

successive value of S in an inexpensive incremental way. Our technique avoids repeated calculation of ****** by

i. establishing the following four invariants on entry to the loop:

The system establishes the invariants using a loop combining transformation called stream processing (see Paige, Koenig[82],Goldberg,Paige[84]).

ii. maintaining these invariants just after they are spoiled by the modification to S within the loop. The maintenance code is called 'difference code' and is generated by RAPTS according to a chain rule.

iv. replacing the computation ****, made redundant within the loop, by the variable new.

Based on structural properties of these eight invariants, RAPTS automatically determines that the cumulative cost of establishing and maintaining them is O(***) with respect to a set theoretic complexity measure. It then be determined that the whole procedure runs in O(***) steps.

Our work in Finite Differencing goes back to Paige,Schwartz[77] and includes Paige[79,83,84,84], Koenig,Paige[81],Paige,Koenig[82].

APPENDIX II. RAPTS EXAMPLES

Below are examples of code generated automatically by the prototype L1 compiler from the given specifications using RAPTS. The first and easiest derivation is of the graph reachability problem. Next is the derivation of a live statement analysis algorithm. The third example shows how the fast Bernstein and Beeri attribute closure algorithm can be compiled from a simple problem specification. The final example is a simple constant propagation. Throughout this session with only one exception manual intervention is required only to supply the names of variables introduced by transformations. The one exception is in the constant propagation example where the property of monotonicity could not be deduced within the current implementation (a practical rather than a theoretical shortcoming).

EXAMPLES

1. Graph Reachability

```
program
     program tclose;
      read ( e , w ) ;
      print ( the s: w subset s | e[s] subset s minimizing #s ) ;
    end program ;
```

```
program
    program reach ;
      read ( e , w ) ;
      a := { } ;
      b := { } ;
      ( forall x27 in a )
        ( forall x26 in e { x27 } | x26 notin b )
          b with := x26 ;
        end forall ;
      end forall ;
      d := { } ;
      c := { } ;
      ( forall x29 in w )
        if x29 notin a then
          d with := x29 ;
        end if ;
        c with := x29 ;
      end forall ;
      ( forall x29 in b | x29 notin c )
        if x29 notin a then
          d with := x29 ;
        end if ;
        c with := x29 ;
      end forall ;
      ( while exists x24 in d )
        ( forall x28 in e { x24 } | x28 notin b )
          if x28 notin w then
            if x28 notin a then
              d with := x28 ;
            end if ;
            c with := x28 ;
          end if ;
          b with := x28 ;
        end forall ;
        if x24 in c then
          d less := x24 ;
        end if ;
        a with := x24 ;
      end while ;
      print ( a ) ;
    end program ;
```

2. Live Code Analysis

```
program
 program useless ;
      assume oneone(instof);
      assume onemany(iuses);
      assume manyone(compound);
      assume disjoint(range instof,range compound);
      read ( instof , usetodef , iuses , compound , crit ) ;
      print ( the live: crit subset live | (instof [ usetodef [ iuses [
        live ] ] ] + compound [ live ] ) subset live minimizing #live) ;
 end ;

program
    program useless ;
      assume oneone ( instof ) ;
      assume onemany ( iuses ) ;
```

```
assume manyone ( compound ) ;
assume disjoint ( range instof , range compound ) ;
read ( instof , usetodef , iuses , compound , crit ) ;
a := crit ;
d := { } ;
c := { } ;
e := { } ;
( forall x3 in a )
  ( forall x2 in iuses ( x3 ) )
    ( forall x7 in usetodef ( x2 ) | x7 notin c )
      d with := instof ( x7 ) ;
      c with := x7 ;
    end forall ;
  end forall ;
  if compound ( x3 ) notin e then
    e with := compound ( x3 ) ;
  end if ;
end forall ;
g := { } ;
f := { } ;
( forall x10 in d )
  if x10 notin a then
    g with := x10 ;
  end if ;
  f with := x10 ;
end forall ;
( forall x10 in e )
  if x10 notin a then
    g with := x10 ;
  end if ;
  f with := x10 ;
end forall ;
( while exists x1 in g )
  ( forall x4 in iuses ( x1 ) )
    ( forall x7 in usetodef ( x4 ) | x7 notin c )
      if instof ( x7 ) notin a then
        g with := instof ( x7 ) ;
      end if ;
      f with := instof ( x7 ) ;
      c with := x7 ;
    end forall ;
  end forall ;
  if compound ( x1 ) notin e then
    if compound ( x1 ) notin a then
      g with := compound ( x1 ) ;
    end if ;
    f with := compound ( x1 ) ;
    e with := compound ( x1 ) ;
  end if ;
  if x1 in f then
    g less := x1 ;
  end if ;
  a with := x1 ;
end while ;
print ( a ) ;
end ;
```

3. Attribute Closure

```
program
 program aclose ;
      read ( x , f ) ;
      print (the s: x subset s | forall y in domain f |
            y subset s impl f(y) subset s  minimizing #s ) ;
    end ;

program
    program aclose ;
      read ( x , f ) ;
      a := x ;
      h := { } ;
      ( forall x21 in domain f )
        ( forall x20 in x21 )
          h ( x20 ) with := x21 ;
        end forall ;
      end forall ;
      c := { } ;
      ( forall x11 in domain f )
        ( forall x10 in x11 )
          if x10 notin a then
            c ( x11 ) := ( c ( x11 ) ? 0 ) + 1 ;
          end if ;
        end forall ;
      end forall ;
      g := { } ;
      e := { } ;
      ( forall x16 in domain f )
        if c ( x16 ) = 0 then
          ( forall x19 in f ( x16 ) | x19 notin e )
            if x19 notin a then
              g with := x19 ;
            end if ;
            e with := x19 ;
          end forall ;
        end if ;
      end forall ;
      ( while exists x9 in g )
        ( forall x13 in h ( x9 ) )
          if x13 in domain f then
            if ( not c ( x13 ) = 0 ) and c ( x13 ) = 1 then
              ( forall x19 in f ( x13 ) | x19 notin e )
                if x19 notin a then
                  g with := x19 ;
                end if ;
                e with := x19 ;
              end forall ;
            end if ;
          end if ;
          c ( x13 ) - := 1 ;
        end forall ;
        if x9 in e then
          g less := x9 ;
        end if ;
        a with := x9 ;
      end while ;
      print ( a ) ;
    end ;
```

4. Constant Propagation

```
program
program const;
 read(assign,usetodef,compute);
 print(the const: empty subset const |
    (const = {s in assign | (forall t in (domain usetodef){s)
              | ((forall x in usetodef([s,t]) | x in const) and
                #{compute(x): x in usetodef([s,t]) * const} <= 1)}))
      minimizing # const);
end;
```

```
program
    program const ;
      read ( assign , usetodef , compute ) ;
      out := empty ;
      m := { } ;
      ( forall x45 in domain usetodef , x46 in usetodef { x45 } )
        m { x46 } with := x45 ;
      end forall ;
      b := { } ;
      ( forall [ x19 , x20 ] in usetodef )
        if x20 notin out then
          b { x19 } + := 1 ;
        end if ;
      end forall ;
      h := { } ;
      g := { } ;
      ( forall [ x30 , x31 ] in usetodef )
        if x31 in out then
          if compute { x31 } notin g { x30 } then
            h { x30 } + := 1 ;
            g { x30 } with := compute { x31 } ;
          end if ;
        end if ;
      end forall ;
      j := { } ;
      ( forall [ x41 , x40 ] in { domain usetodef } )
        if h { x41 , x40 } > 1 then
          j { x41 } + := 1 ;
        end if ;
      end forall ;
      d := { } ;
      ( forall [ x26 , x25 ] in { domain usetodef } )
        if b { x26 , x25 } > 0 then
          d { x26 } + := 1 ;
        end if ;
      end forall ;
      l := { } ;
      k := { } ;
      e := { } ;
      ( forall x29 in assign )
        if d { x29 } = 0 then
          if j { x29 } = 0 then
            if x29 notin out then
              l with := x29 ;
            end if ;
            k with := x29 ;
```

```
    end if ;
    e with := x29 ;
  end if ;
end forall ;
( while exists x2 in l )
  ( forall x21 in m ( x2 ) )
    if x21 in ( domain usetodef ) and b ( x21 ) = 1 then
      if x21 ( 1 ) in assign then
        if d ( x21 ( 1 ) ) = 0 then
          if j ( x21 ( 1 ) ) = 0 then
            if x21 ( 1 ) notin out then
              l less := x21 ( 1 ) ;
            end if ;
            k less := x21 ( 1 ) ;
          end if ;
          e less := x21 ( 1 ) ;
        elseif d ( x21 ( 1 ) ) = 0 + 1 then
          if j ( x21 ( 1 ) ) = 0 then
            if x21 ( 1 ) notin out then
              l with := x21 ( 1 ) ;
            end if ;
            k with := x21 ( 1 ) ;
          end if ;
          e with := x21 ( 1 ) ;
        end if ;
      end if ;
      d ( x21 ( 1 ) ) - := 1 ;
    end if ;
    b ( x21 ) - := 1 ;
  end forall ;
  ( forall x35 in m ( x2 ) )
    if compute ( x2 ) notin g ( x35 ) then
      if x35 in ( domain usetodef ) then
        if h ( x35 ) = 1 - 1 then
          if x35 ( 1 ) in e then
            if j ( x35 ( 1 ) ) = 0 then
              if x35 ( 1 ) notin out then
                l less := x35 ( 1 ) ;
              end if ;
              k less := x35 ( 1 ) ;
            elseif j ( x35 ( 1 ) ) = 0 - 1 then
              if x35 ( 1 ) notin out then
                l with := x35 ( 1 ) ;
              end if ;
              k with := x35 ( 1 ) ;
            end if ;
          end if ;
          j ( x35 ( 1 ) ) + := 1 ;
        elseif h ( x35 ) = 1 then
          if x35 ( 1 ) in e then
            if j ( x35 ( 1 ) ) = 0 then
              if x35 ( 1 ) notin out then
                l less := x35 ( 1 ) ;
              end if ;
              k less := x35 ( 1 ) ;
            elseif j ( x35 ( 1 ) ) = 0 + 1 then
              if x35 ( 1 ) notin out then
                l with := x35 ( 1 ) ;
              end if ;
```

```
                k with := x35 ( 1 ) ;
              end if ;
            end if ;
            j ( x35 ( 1 ) ) - := 1 ;
          end if ;
        end if ;
        h ( x35 ) + := 1 ;
        g ( x35 ) with := compute ( x2 ) ;
      end if ;
    end forall ;
    if x2 in k then
      l less := x2 ;
    end if ;
    out with := x2 ;
  end while ;
  print ( out ) ;
end ;
```

Geometric Modeling: Software Research and Development*

Chanderjit L. Bajaj
Department of Computer Science
Purdue University
West Lafayette, IN 47907.

May 21, 1989

Abstract

The project's research emphasis is on the computational and mathematical infrastructure, needed to support the software development of a geometric modeling system. This is a software system which allows for the efficient creation and manipulation of concise boundary representations of curved solid models of physical objects. The geometric coverage includes algebraic curves and surfaces of arbitrary degree, and allows both the implicit and rational parametric representations with both power and Bernstein polynomial bases. Utilities are provided for automatic conversions between the various polynomial representations. Furthermore, a graphical user interface allows creation of Bezier control polygons and polyhedra, for efficient design of Bezier curve segments and Bezier surface patches. Geometric operations include boolean set operations, offsets, sweeps, solid decompositions, and wireframe flesh via interpolation. Graphical display facilities include quick wireframe plot and redisplay of the boundary representation, three dimensional transformations of the solid, and high resolution color rendering. The geometric modeling system is being implemented in Common Lisp, Fortran and C on a combination of Symbolics 3650, HP - 370SRX Turbo, and SUN 4-110 platforms.

*Supported in part ONR contract N00014-88-K-0402.

1 Background

In the summer of 1986 plans were made for developing a geometric modeling system for the efficient creation and manipulation of accurate computer models of solid physical objects. A primary goal since then, is to accurately model the boundary of rigid physical objects with algebraic surface patches. The focus is on using the lowest degree surface patches which satisfy the design constraints, since lower degree surfaces lends itself to faster computations in geometric design operations as well as in tasks such as computer graphics display, animation, and various physical simulations. The project's research emphasis is on the computational and mathematical infrastructure, needed to support the software development of this geometric modeling system.

Geometric Coverage: We focus on the use of low degree, implicitly defined, algebraic surfaces in three dimensional space $\mathbf{R}^3$. A real algebraic surface S in $\mathbf{R}^3$ is implicitly defined by a single polynomial equation $f(x, y, z) = 0$, where coefficients of f are over the real numbers $\mathbf{R}$. A real algebraic space curve can be defined by the intersection of two real algebraic surfaces and implicitly represented as a pair of polynomial equations ($f_1(x, y, z) = 0$ and $f_2(x, y, z) = 0$) with coefficients again over the real numbers $\mathbf{R}$. In modeling the boundary of physical objects it suffices to consider only space curves defined by the intersection of two algebraic surfaces. Space curves in general are defined by the intersection of several surfaces. A *rational* algebraic space curve can also be represented by the triple $(x = G_1(s), y = G_2(s), z = G_3(s))$, where G_1, G_2 and G_3 are rational functions in s. Whenever we consider the special case of a rational space curve, we assume that the curve is smooth and only singly defined under the parameterization map, i.e., each triple of values for (x, y, z), corresponds to a single value of s.

Why algebraic surfaces ? Manipulating polynomials, as opposed to arbitrary analytic functions, is computationally more efficient. Furthermore algebraic surfaces provide enough generality to accurately model almost all complicated rigid objects. Also, algebraic curves and surfaces lend themselves very naturally to the difficult computational problem of physical object design.

Why implicit representations ? Most prior approaches to geometric and soid modeling, have focused on the parametric representation of surfaces. Contrary to major opinion and as we exhibit through our research, implicitly defined surfaces are also very appropriate. Additionally, while all algebraic surfaces can be represented implicitly, only a subset of them have the alternate parametric representation, with x, y and z given explicitly as rational functions of two parameters. Furthermore, implicit algebraic curves and surfaces have compact storage representations and form a class which is closed under most common operations required by a geometric modeling system.

2 Research Issues, Objectives & Directions

- Develop computational techniques using, algebraic geometry and numerical approximation theory to eliminate bottlenecks in geometric modeling operations. These include robust Boolean set operations (union, intersection, etc.), solid decompositions, offsets, envelopes and sweep computations. Accuracy and robustness are two of the most pressing technical issues in geometric modeling.

- Develop and implement efficient "modular" algorithms for algebraic curve and surface parameterization, implicitization and singularity resolution. Utilize efficient methods of Chinese remaindering, Hensel lifting and multivariate interpolation.

- Develop graphical user interfaces for easy editing of geometric object information. Includes three dimensional transformations such as translation, rotation, scaling, etc.

3 Approaches & Progress

- **Algebraic Boundary Model Creation** – an editing toolkit.

  This package is in CLisp and Fortran on the Symbolics and is one of continual growth, [4, 11, 12, 16, 19, 21, 22, 23]. Capabilities are (a) allows quick wireframe plot and redisplay of the curved solid boundary data structure for arbitrary algebraic surfaces. Has a robust surface-surface intersection routine with calls to a Fortran SVD subroutine. Also a revamped makesolid routine to produce an internal form of the boundary description for solid manipulation routines (boolean operations, triangulation,...). (b) produces a complete boundary description of "offsets" of points, line segments, angles (two line segments),.... From there the plot capabilities of (a) take over Takes input, one, two, three ....points and an offset radius. 3d solid transformation routines i(scaling, translation, rotation) are also implemented and used to derive the boundary description. (c) handles "extrudes" and "curved solids of revolution" on the same lines as (b). Study of robustness issues of curved model reconstruction and display, via symbolic reasoning, is underway. Benefits all projects below which manipulate solids with boundary descriptions. (Summer project for undergrad: Implement a color render program of the solid boundary description for the HP graphics workstation.)

  Current Programmer: Steven Klinkner

- **Robust Polyhedra Triangulation** – robust modeling operation using the topological reasoning paradigm.

  This package is in CLisp on the Symbolics and is near completion [10]. Takes as input an arbitrary simple polyhedra, in a modified Karasick's external boundary description and produces a convex decomposition/ triangulation of the polyhedra. All the pieces are returned in the boundary description. An algorithm was developed where

the theoretical bound of Chazelle's old 1984 algorithm was improved by a factor of $O(N^2/logN)$, where N = number of reflex edges. Chazelle recently informed us that he had shaved off the additional $O(logN)$ factor by a different method. Implementation was done to better understand the issues of robustness. Rewrote and corrected some of Karasick's robust classification routines. Developed a robust plane-sweep algorithm for detecting edgecycle loops. Has an input and display interface from the editing toolkit above, as well as from S-Geometry. Color shaded pictures can also be produced. Should prove useful in calculating volumetric properties of solid models and in interfacing simple finite element programs. Next step: Robust Triangulation of Curved Solids.

Current Programmer: Tamal Dey

- **Hermite Interpolation with Algebraic Surfaces** – automatic surface generation.

  This package is in CLisp and C on a HP color graphics workstation. Version One completed in February end [15]. Currently working on Version Two with a better user interface and new features. Takes as input: points and curves in space together with "normal" directions and produces a family of the lowest degree, algebraic surface which "smoothly" interpolates the points and curves. Nonsingularity and convexity constraints, as well numerical conditioning issues are satisfied by doing computations with polynomials in Bernstein basis (as opposed to the traditional power basis). Currently uses the Macsyma routine for linear system solutions over integral domains. Has a textual input interface and uses the graphics workstation hardware for color display. Should prove useful in fleshing curved wireframes, for smooth meshing with low degree surfaces, generating blending and joining surfaces and for constructing low degree curved finite elements.

  Current Programmer: Insung Ihm

- **Package for Solving Systems of Polynomial Equations** – all roots solver.

  This package is in CLisp and C on a SUN 4 and is a long term effort [1, 2, 3, 6, 8, 9, 13, 14]. The goal is to have both a robust and efficient set of routines to construct compact representations for all the roots of a general system of multivariate polynomial equations. For 0 dimensional solutions , approximate real solutions are obtained within "epsilon" neighborhoods of the true solutions. For k dimensional solutions in n space, a hypersurface in $k+1$ dimensional space is generated, together with points on the true solutions expressed as rational functions of points on the hypersurface. Methods are based on Sylvester and Macaulay resultants and subresultants as well as symbolic parameterization routines. A very fast Sylvester routine, based on Chinese remaindering, is implemented and a similar implementation for Macaulay's resultant is currently underway. Global parameterization routines for upto degree three hypersurfaces have also been implemented. Currently uses the Macsyma routine for univariate polynomial real root solving, and curve tracing routines to display the zero, one and

two dimensional solutions. The package has an interactive user interface to enter equations and select different ways of solving and displaying solutions. Provides a basic mathematical package of polynomial manipulation routines for varied applications.

Current Programmer: Andrew Royappa

- **Power series factorizations and Pade approximations** – analyzing curve and surface singularities.

  This package is in CLisp on the Symbolics and is complete for curves [5, 7]. The algorithms for surfaces are simultaneously being developed and implemented. Takes implicit algebraic curves and surfaces as input and produces a power series parameterization for all the branches at a singularity (curve branches about a singular point, and surface branches about a singular curve). Further Pade' rational approximants can be computed for the power series parameterizations. Has an interactive, textual user interface, where different degrees of approximation can be specified. The output is both textual and graphical, displaying the original and approximated branches of the curve and surface. The algorithms are based on Hensel lifting of power series, yielding Newton and Weierstrass factorizations. The Pade' routines are based on the Brent, Gustavson and Yun method of using the extended GCD algorithm to solve Toeplitz matrix computations. Provides the essential routines for constructing a piecewise rational approximation of any curve or surface. Projected usefulness in modeling and graphics.

  Current Programmer: Chanderjit Bajaj

- **Compliant Path Planner** – generating contact paths for a curved object with fixed orientation.

  This package is in CLisp on the Symbolics and is complete for a planar curved model, moving with fixed orientation, and in continuous contact with other static planar curved models [17]. The implementation for solid models is pending [18, 20]. The method is based on the convolution computation, made efficient with simple "paint" heuristics. Has a menu driven, graphical interface to specify and display planar model descriptions. The planar models currently are made of piecewise circular arcs and straight lines. The compliant path is demonstrated by a graphical animation of a planar model moving in continuous contact with the fixed curved models. Next Step: Besides upgrading this to three dimensions, we hope to interface this planner with Newton.

- **Multiple Object Motion Coordination** – path generation through simulation.

  This package is in CLisp on the Symbolics and is first being programmed for coordinating the simultaneous collision free motion of homogenous simple discs in the plane. The approach is Voronoi based, where at each time step a disc considers only its voronoi neighbors as potential collision threats. A dynamic planar Voronoi diagram is being implemented. A static planar Voronoi diagram is already complete. The velocity and

acceleration of the discs is handled by the dynamic equations of Newton. Next Step: To move onto three dimensions.

Current Programmer: Bill Bouma

4 Miscellaneous Topics

Grand Challenge:

The accurate re-design of the exterior geometry of the space shuttle Discovery or carrier Enterprise in three[1] hours, or less.

Research Transitions:

Geometric modeling tools necessarily find a wide range of design applications in large volume, manufacturing industries such as Boeing, General Motors, Ford, General Dynamics, Department of Defense, etc. These tools are being used for the design of the exterior geometry of airplanes, automobiles, rockets, ships, space shuttles, ... as well as for the numerous parts (motors, engines, drive-shafts, etc) that they consist of. More recently however, they are increasingly being used by personnel involved in the physical and bio-medical sciences. Examples abound in artifical limb design, crystallography, genetic research, pharmacuetical research, and more.

Traditionally, there has been a large lag time between university research and its successful incorporation for specific enhancements in industrial products. However, with the close proximity and immediate relevance of geometric modeling research to industrial applications, this gap should definitely be bridged. Immediate possibilities are through joint industry-university conferences, and contractual university research funded by industry, while long term goals may be met by industry sponsored, university courses and laboratories.

Technological Impacts:

Our current research and experimentation hardware consists of three Symbolics 3650 color workstations (about 3 MIPs but excellent software development environment); a HP - 370SRX Turbo color graphics workstation (about 4 MIPs with graphics accelerators with hidden surface removal and display transformations such as rotate, translate and zoom, in hardware or firmware); a color Sun 4-110 (about 7 MIPs for quick computations, and useful in experimenting with floating point and polynomial arithmetic); and access to an Alliant FX-80 (a four processor number cruncher).

Integration of the software (and hardware) environment of these machines posed a big challenge and many man months were spent achieving a certain level of compatibility. These problems seem to have been somewhat resolved by the recent announcements of graphics superworkstations (e.g. Ardent Titan or the Silicon Graphics IRIS 4D/240 GTX), which combine MIPS power with graphics pipelines, and seem to be targeted at geometric modeling

[1] Estimated as the maximum single, continuous sitting time of a sophisticated designer

and simulation projects such as ours. These new breed of machines shall definitely lead to enhanced capabilities for research and experimentation in geometric modeling with high degree algebraic surfaces.

5 PUBLICATIONS

1. S. Abhyankar and C. Bajaj, "Computations with algebraic curves", *Proc. of Intl. Symposium on Symbolic and Algebraic Computation*, (ISSAC88), *Lecture Notes in Computer Science*, Springer-Verlag, (1988), accepted for publication.

2. S. Abhyankar and C. Bajaj, "Automatic parameterization of rational curves and surfaces III: Algebraic plane curves", *Computer Aided Geometric Design*, **5**, (1988), 309–321.

3. S. Abhyankar and C. Bajaj, "Automatic parameterization of rational curves and surfaces IV: Algebraic space curves", presented at the *1987 SIAM Conference on Applied Geometry*, Albany, NY. (Accepted for publication in *ACM Transactions on Graphics.*)

4. C. Bajaj, "Geometric modeling with algebraic surfaces", *The Mathematics of Surfaces III*, (D. Handscomb, ed.), Oxford University Press, (1989), to appear.

5. C. Bajaj, "Approximation methods for algebraic curves and surfaces", Technical Report, CAPO-88-36, Purdue University, West Lafayette, IN, November 1988. Invited paper at the workshop on *Algorithmic Aspects of Geometry and Algebra*, Cornell MSI, Ithaca, NY. Final version expected to appear in the workshop proceedings.

6. C. Bajaj, "Mathematical techniques in solid modeling", *Proc. of International Conference on Computer Integrated Manufacturing*, RPI Troy, NY, (1988), 290–295.

7. C. Bajaj, "Local parameterization, implicitization and inversion of real algebraic curves", *Proc. of the AAECC-7*, Toulouse, France (1988), to appear.

8. C. Bajaj, "Quadric and cubic hypersurface parameterization", Technical Report, CAPO-89-14, Purdue University, West Lafayette, IN, April 1989.

9. C. Bajaj, J. Canny, T. Garrity and J. Warren, "Absolute factorization of bivariate polynomials, *Proc. of ISSAC-89*, Portland, Orgeon (1988), to appear.

10. C. Bajaj and T. Dey, "Robust decompositions of polyhedra", Technical Report, CAPO-88-44, Purdue University, West Lafayette, IN, December 1988, (revised May 1989).

11. C. Bajaj, W. Dyksen, C. Hoffmann, E. Houstis, T. Korb, and J.R. Rice, "Computing about physical objects", *Proc. 12th World Congress on Scientific Computing, IMACS*, 4 (1988), 642–644.

12. C. Bajaj, W. Dyksen, C. Hoffmann, E. Houstis, T. Korb and J. Rice, "Interface structures I: Abstract structures for computing about physical objects", Technical Report, CAPO-88-25, Purdue University, West Lafayette, IN, June 1988.

13. C. Bajaj, T. Garrity and J. Warren, "On the applications of multi-equational resultants", Technical Report, CAPO-88-39, Purdue University, West Lafayette, IN, November 1988.

14. C. Bajaj, C. Hoffmann, J. Hopcroft and R. Lynch, "Tracing surface intersections", *Computer Aided Geometric Design*, **5**, (1988), 285–307.

15. C. Bajaj and I. Ihm, "Hermite interpolation of rational space curves using real algebraic surfaces", *Proc. of 5th Annual ACM Symposium on Computational Geometry*, Saarbrucken, West Germany, (1989), accepted for publication.

16. C. Bajaj and M. Kim, "Convex hull of curved objects bounded by algebraic curves", presented at the *1987 SIAM Conference on Applied Geometry*, Albany, NY. (Accepted for publication in *Algorithmica*.)

17. C. Bajaj and M. Kim, "Generation of configuration space obstacles: The case of moving algebraic curves", *Algorithmica*, **4**, 2, 1989, 157-172.

18. C. Bajaj and M. Kim, "Compliant motion planning with geometric models", *Proc. of 3rd ACM Symposium on computational Geometry*, Waterloo, Canada, (1987), 171–180. (Updated version with title "Generation of configuration space obstacles: The case of moving algebraic surfaces", accepted for publication in *International Journal of Robotics Research*, (1989).)

19. C. Bajaj and M. Kim, "Algorithms for planar geometric models", *Proc. of the Fifteenth Intl. Colloquium on Automata, Languages and Programming*, (ICALP 88), *Lecture Notes in Computer Science*, Springer-Verlag, 317, (1988), 67–81.

20. C. Bajaj and M. Kim, "Generation of configuration space obstacles: The case of moving spheres", *IEEE Journal of Robotics and Automation*, **4**, 1, (1988), 94–99.

21. C. Bajaj and M. Li, "Geometric optimization and DP-completeness", *Discrete and Computational Geometry*, **4**, 1, (1989), 3–13. (Abstract appears in *Zentralblatt fur Mathematik*.)

22. C. Bajaj, M. Wu and C. Liu, "A face area evaluation algorithm for solids", *Computer Aided Design*, **20**, 2, (1988), 75–82.

23. J. Johnstone and C. Bajaj, "On the sorting of points along an algebraic curve", Technical Report, CAPO-88-37, Purdue University, West Lafayette, IN, November 1988.

MODELING PHYSICAL OBJECTS

Christoph M. Hoffmann
Computer Science Department
Purdue University

Abstract

The research develops the infrastructure necessary for comprehensive, user-friendly software systems that are capable of modeling and analyzing physical objects and systems of physical objects. The focus of the work is on the following major areas:

- The logical foundations required to implement, without failure, the supporting geometric operations in the face of limited precision arithmetic and uncertainties of position.
- The mathematical foundations of object representations, with specific emphasis on efficiency, robustness, and accuracy.
- The development of conceptual primitives to support the design process and to interface diverse mathematical models analyzing physical properties in a variety of contexts.

The project builds software tools and experimental systems that assess the viability of our ideas. Experience shows that our ideas are productive, and the feedback from the community indicates that this work is timely and of value.

Contents

1 Background

This work began during a two-year visit at Cornell University 1984–86, as a collaborative effort with John Hopcroft. I had worked with John before, so we had a history together, and an appreciation of each other's abilities. My prior work had been in graph isomorphism and computational group theory. The narrowness of the computer-science community interested in this subject convinced me that I should look for a broader, and more applied, area of work. So, I came to Cornell at a perfect point in time.

1.1 Context of the Project

Science and manufacturing technology are currently undergoing a major change whose thrust it is to computerize all constituting processes and automate design, analysis, and evaluation. Sign posts of this restructuring are the name change of the National Bureau of Standards, the inception of major research initiatives, such as DARPA's DICE project, and the increasing interest in the computational paradigm by the natural sciences.

This ongoing restructuring is made possible by the wider availability of larger and faster computers that are, in effect, opening up new dimensions in problem scale and detail that can be effectively contemplated. It necessitates an interdisciplinary effort to devise proper models of physical objects that are amenable to interrogation and modification through computing. Our effort is at the center of computer aided design, computer aided manufacture, robotics, and computational science.

The main thrusts of our work are research in geometric and solid modeling, and research in automating the application of these models in a variety of endeavors including the simulation of physical phenomena and their computational analysis.

- In solid modeling, we investigate all levels of the design and implementation process, including problems arising at the conceptual design level, problems to be solved when extending the geometric capabilities, and problems dealing with the foundational issues raised by implementations.
- In physical simulation, we are refining the Newton system developed jointly with Cornell University, and concentrate on expanding the physical coverage by studying interface problems that integrate this system with other, existing simulation and analysis systems of complementary capability.

Since its inception five years ago, this work has produced a rich variety of results and prototypes. Past and present efforts are coextensive with related efforts, most notably the work done by John Hopcroft at Cornell. This is most appropriate given the nature and magnitude of the work to be done.

2 Research Objectives

The goal of this work is to create a science base for computer representation, analysis and manipulation of models of physical objects, and to develop the infrastructure necessary to give wide applicability to the insights and techniques developed in the course of the project. At this time, two specific foci are in the foreground:

1. Develop and broaden geometric and solid-modeling capabilities.
2. Develop and broaden tools for modeling and simulating systems of physical objects.

Each of these foci splits into several subefforts that conceptualize current problems and how they can be overcome. In geometric and solid modeling, the following objectives are pursued:

1. Investigate what makes the substrata unreliable on which geometric modeling is implemented, and develop ways to make it reliable.
2. Develop the algorithmic and mathematical infrastructure needed to enlarge the geometric coverage of modelers and make modelers more efficient.
3. Develop good user interface languages ultimately resulting in increased productivity in engineering design.

In simulation and modeling of physical systems the following objectives are pursued:

1. Investigate the interaction between geometric shape and physical behavior.
2. Investigate modalities and design methods to accomodate changing modalities.
3. Integrate the computational treatment of different physical aspects, such as motion, heat transfer, stress and vibration.

The two areas of research are portrayed in separate sections.

3 Research Issues in Geometric and Solid Modeling

The geometric and solid modeling research addresses the computer science aspects of how to represent, manipulate, and analyze the shape of physical objects by computer. The work focuses on three subareas: substrata issues, infrastructure issues, and user-interface issues.

3.1 Problems in Geometric and Solid Modeling

Substrata

It is a widely recognized fact that most geometric and solid modeling systems fail under certain conditions. Typically, for objects with surface elements that are almost, but not quite,

coincident, valid inputs to a modeling system may fail to produce valid results and could even crash the system. Less widely recognized is that these problems ultimately root in the fact that the modeler's infrastructure of algorithms has been designed with infinite precision arithmetic in mind, but is typically implemented, for efficiency reasons, using fixed precision arithmetic, i.e., floating point numbers, e.g., Hoffmann, Hopcroft, and Karasick (1987). In consequence, certain computations from which to deduce symbolic geometric facts, e.g., incidence or nonincidence, are inconclusive due to insufficient precision. These inconclusive results must be interpreted, and from them deductions must be made based on incomplete information.

Infrastructure

Techniques are needed to represent, modify, and interrogate objects whose shape elements are algebraics of unrestricted degree. This is a bold undertaking that generalizes at once all previous approaches to solid modeling. For example, free form surface design concentrates on working with specific classes of (parameterizable) algebraic surfaces.

Previous work in this direction has been stymied by severe problems arising when dealing with high-degree algebraics. For this reason, most modeling systems drastically restrict the types and degrees of the allowed surface elements. Our infrastructure research makes use of all available successful techniques, including numerical methods, symbolic computation, and differential geometric techniques, in order to obtain the best results possible. This pragmatism is absolutely essential if the goals of accuracy, efficiency, and robustness are to be attained.

User-Interfaces

Effective use of solid modelers currently requires specialists with much training. There is a need to make these systems accessible to the nonspecialist. Computer science should be able to make sophisticated contributions in this area, given the deep insights the field has gained in programming language design.

3.2 Research Approach to Geometric and Solid Modeling

Assume that we have to make a decision based on unreliable numeric data. If we are dealing with complex geometric objects, such as the representation of solids, the decision to be made will depend on how we decided other uncertain computations during earlier parts of the computation. This interdependence of decisions is not easily recognized algorithmically. Failing to recognize it, however, makes it likely that we make *inconsistent* decisions which could crash the algorithm. This is an important research topic widely acknowledged to be of critical importance. Our work approaches the substrata problems as follows:

1. Develop a clear understanding of the extent of the problem in specific applications such as Boolean operations on solids. We are in the process of completing the tools needed for this work, a dual-mode modeler capable of doing the same operation both in floating point and in exact rational arithmetic, using identical algorithms.

2. Investigate the logical complexity of reasoning about the consequences of the numerical decisions. Past work has developed the reasoning paradigm, current work will extend it.

We use algebraic methods, numerical methods, and methods from differential geometry. The algebraic methods investigate how to apply techniques from algebraic geometry, such as desingularization, and how to make computations such as Grobner bases more effective. An important criterion here is that the methods should not involve excessive or intractable computations.

The numerical approach seeks ways to increase accuracy and geometric coverage by reformulating problems in higher dimensions. A dimensionality paradigm has been formulated, and its utility is currently under investigation. The objective here is to reduce algebraic degrees by introducing more variables.

The differential approach looks at a variety of projection methods, seeking to determine which classical techniques have potential in geometric and solid modeling. We restrict this work for now to these specific problems:

1. Given a point p and a surface in three space, determine the distance of p from the surface, and determine the projection of the point to a surface point q of minimum distance.

2. Given a space curve and a surface, trace the space curve and simultaneously its projection onto the surface. Here, each point p on the curve is projected to a point q of minimum distance on the surface.

Effective design seems to require a notion of "feature". Proper definition of the concept, and an elucidation of the spatial interaction of different features, is a thorny problem that provokes intense discussions in workshops and conferences. No accepted definitions have emerged yet, and this situation may persist for some time to come. We plan to approach the problem from a different perspective: Given a particular object in full detail, "approximate" it by deleting detail features that are unimportant to the overall design. Thus, we attempt to derive a hierarchy of shapes, each progressively less detailed.

To approach this problem and give a satisfactory solution requires experimentation, and we have the necessary tools in place to do this. We plan to examine several complex designs, analyze their features, give a formal feature definition, and devise an approximation algorithm. The results of this algorithm must then be inspected, and judged as to their satisfactoriness. Unsatisfactory approximations can then be analyzed and traced to possible flaws in the feature definition or to unexpected interactions between features.

3.3 Progress in Geometric and Solid Modeling

Past research has isolated the sources of this difficulty in the polyhedral domain. As reported in Hoffmann, Hopcroft and Karasick (1988), the difficulties are traceable to floating point arithmetic impacting logical conclusions drawn, such as vertex/plane incidence. The paper also gives a paradigm for approaching this problem and solving it using symbolic reasoning to ensure consistency. A separate paper, Hoffmann (1989a), surveys the problem in the larger context of geometric computations by computer, and contrasts our approach with others proposed by the field.

We are implementing an experimental modeler which will serve as a test bed for analyzing which geometric errors can be tolerated and which ones are fatal. The modeler design pays special attention to the arithmetic problem and has two modes of operation, one, in which exact arithmetic is used, and another one using floating-point arithmetic. Both versions work with identical algorithms. The modeler will be used as follows:

1. Run the floating point version until a failure has been encountered.

2. With identical input, run the exact arithmetic version.

3. If the exact arithmetic version also fails, then we have uncovered an algorithmic error. Otherwise, the failure is a consequence of the limited precision arithmetic.

4. Note that by using the results of the exact arithmetic version we can classify the type of failure.

A two-dimensional version is already operational, and shows that some of the types of failure cited in the literature are, in fact, not failures due to precision problems, but are programming errors. The completed 3D version will give us a platform to systematically investigate the usefulness of solutions proposed by us and others.

Symbolic algebraic methods have been successfully applied to a variety of problems, including tracing plane algebraic curves through arbitrary singularities, Bajaj, Hoffmann, Hopcroft, and Lynch (1988). Moreover, for specific problems such as the elimination of variables from systems of algebraic equations, we have developed what we believe is the fastest known method. We can successfully tackle problem sizes that cannot be solved by any other approaches that have been implemented.

There are many difficult surface operations that one would like to implement but cannot do so because the traditional approach entails intractable symbolic computations. Many of these operations become almost trivial when reformulated in higher dimensional spaces. These operations include surface offsets, needed in numerically controlled machining, Voronoi surfaces, needed to precisely formulate certain blending surfaces, and blending surfaces that must satisfy special constraints such as circularity of cross section. That is, the derived surface is formulated as a set of equations with more than three variables, and this multi-equational representation is used directly. In Hoffmann (1988) we demonstrate that curves of algebraic degree well over 100 can be traced with normal double-precision floating point arithmetic, to an accuracy of ten significant decimals.

Work is underway to examine the various surface interrogations of importance, and to assess how they might be restructured to work with the higher dimensional version. These methods include subdivision in higher dimensions, as a guaranteed method to localize the various branches; local approximations to the surface without any variable elimination; and distance function computations.

Unlike algebraic methods, techniques from differential geometry are not yet in wide-spread use in geometric and solid modeling. It is not clear why this is the case, but we expect that this situation will change, and we are exploring the utility of differential concepts in solid modeling.

Joint with F.-E. Wolter, we have developed several experimental programs to find projections and track the projection of a curve to a surface. These tools are presently unsatisfactory, but there appear to exist ways to improve them. These ways would modify what is now a classical differential approach, and integrate some algebraic techniques.

4 Research Issues in Physical Modeling

Project Newton develops a highly modularized and extensible system to duplicate the precise behavior of physical objects from their models. The work is done cooperatively with John Hopcroft (Cornell) and should have a major impact on computer science, engineering, and manufacturing.

4.1 The Problem

Simulation and analysis of physical systems is a vast subject in which there has been much work in nearly all branches of science and engineering. Despite this long and illustrious history, we can identify several gaps in the traditional approaches:

1. *Limited Geometry.* Complicated shapes are not normally modeled, and the interaction between shape and consequent physical behavior is relatively unexplored.

2. *Fixed Modality.* Things are either elastically deforming, or they flow plastically. The change from one behavior to the other is not modeled.

3. *No Multiplicity.* Real objects behave and interact with the environment in a multiplicity of ways. They may simultaneously: accelerate, heat up or cool down, vibrate, and so on. Typically, only one aspect is modeled; any interaction between the various aspects is ignored.

While many questions associated with these limitations belong to specific disciplines in science and engineering, there is a computer science component that can and should be investigated. Moreover, the questions are in many respects interdisciplinary.

4.2 Approach to Physical Modeling

The simulation of objects and their physical behavior is based on the geometry of their shapes. From the geometric descriptions, the system formulates automatically the needed mathematical models that describe the laws of possible mechanical motion. As the simulation progresses, the system will reformulate these models as needed; for example, in response to collision, or a disappearing contact between two objects. Both, the automatic model formulation and the automatic model modification are novel aspects of the work.

The original system design is based on Newtonian mechanics. Methods are being explored to overcome the intrinsic limitations of Newtonian mechanics and to increase the scope of phenomena that can be simulated. We refer to this research as extending the *physical* coverage, just as our research in geometric modeling aims at extending the *geometric* coverage.

Extending the physical coverage does not necessarily involve breaking new ground in physics or mechanical engineering, since most of the phenomena we would like to simulate can already be simulated by suitable finite element codes. However, finite element codes are developed for specific physical phenomena in isolation, and we would like to track the phenomena simultaneously. Moreover, these programs are meant to be used by specialists. They have limited geometric capabilities and limited automatic capabilities. It is our aim to interface the Newton system with these codes in such a way that human intervention and problem formulation becomes largely unnecessary. This activity is in part similar to software engineering, in that we wish to combine existing complex software systems with each other without extensively rewriting them.

4.3 Progress in Physical Modeling

A second implementation is now operational, developed in Common Lisp on Symbolics Lisp machines. Its coverage includes geometric shape, rigid body dynamics, control model evaluation, interference detection, and collision simulation.

An experimental interface to finite element codes has been constructed, but it is not yet fully general. First experiments demonstrate that it is possible to extend the Newton system by interfacing it with complementary software packages. Moreover, this interface is across physical machine boundaries, i.e., the Newton system is in the process of being distributed over a network of cooperating machines.

The scientific problems raised by this interface include determining faithfully the initial values entailed by the collision. A possible way to derive the initial conditions for the FEM problem is to first determine the impulses resulting from the collision, and then to assign the appropriate initial velocities to the elements involved. More general paradigms are under investigation.

5 References

Bajaj, C., C. Hoffmann, J. Hopcroft and R. Lynch (1988)
"Tracing Surface Intersections," *Computer Aided Geometric Design 5*, 285–307.

Chandru, V., D. Dutta, and C. Hoffmann (1989)
"On the Geometry of Dupin Cyclides," TR 88-818, Comp. Sci., Purdue University.

Chandru, V., D. Dutta, and C. Hoffmann (1989)
"Variable Radius Blending using Dupin Cyclides," TR 89-851, Comp. Sci., Purdue University.

Chuang, J.-H., and C. Hoffmann (1989)
"On Local Implicit Approximations of Curves and Surfaces," *ACM Trans. Comp. Graphics*, to appear.

Hoffmann, C., (1987)
"Algebraic Curves," in *Mathematical Aspects of Scientific Software*, J. Rice, ed., IMA Volumes in Math. and Applic., Springer Verlag, 101–122.

Hoffmann, C., (1988)
"A Dimensionality Paradigm for Surface Interrogations," TR 88-837, Comp. Sci., Purdue University.

Hoffmann, C., (1989a)
"The Problem of Accuracy and Robustness in Geometric Computation," *IEEE Computer*, to appear.

Hoffmann, C., (1989b)
Geometric and Solid Modeling, An Introduction, Morgan Kaufmann Publishers, to appear August 1989.

Hoffmann, C., and J. Hopcroft (1985)
"Automatic Surface Generation in Computer Aided Design," *The Visual Computer 1*, 92–100.

Hoffmann, C., and J. Hopcroft (1986)
"Quadratic Blending Surfaces," *Comp. Aided Design 18*, 301–307.

Hoffmann, C., and J. Hopcroft (1987a)
"The Potential Method for Blending Surfaces and Corners," in *Geometric Modeling*, G. Farin, ed., SIAM Publications, Philadelphia.

Hoffmann, C., and J. Hopcroft (1987b)
"Simulation of Physical Systems from Geometric Models," *IEEE J. Robotics and Autom. RA-3*, 194–206.

Hoffmann, C., and J. Hopcroft (1987c)
"Geometric Ambiguities in Boundary Representations," *Comp. Aided Design 19*, 141–147.

Hoffmann, C., and J. Hopcroft (1988a)
"Projective Blending Surfaces," *Artif. Intelligence 37*, 357-376.

Hoffmann, C., and J. Hopcroft (1988b)
"Model Generation and Modification for Dynamic Systems from Geometric Data," in *CAD Based Programming for Sensory Robots*, B. Ravani, ed., Springer NATO ASI Series F-50, 481–492.

Hoffmann,C., J. Hopcroft and M. Karasick (1987)
"Robust Set Operations on Polyhedral Solids," Tech. Rept. 723, Comp. Sci., Purdue University.

Hoffmann, C., J. Hopcroft and M. Karasick (1988)
"Towards Implementing Robust Geometric Computations," 4^{th} *ACM Symp. on Comp. Geometry*, 106–117.

6 Results from Prior Naval Support

We summarize the work and accomplishments due to the prior support through contract N00014-86-K-0465, during the period of 8/86 through now.

6.1 Books

1. "Geometric and Solid Modeling", to be published by Morgan Kauffman, San Francisco, July 1989.
2. Editor of "Issues in Robotics," JAI Press, to appear late 1989.
3. Editor of "Algorithmic Aspects of Geometry and Algebra," Springer Verlag; (with E. Kaltofen and C. Yap).

6.2 Papers and Technical Reports

1. "The Potential Method for Blending Surfaces and Corners," in *Geometric Modeling*, G. Farin, ed., 347–365, SIAM 1987.
2. "Simulation of Physical Systems from Geometric Models," *IEEE J. Robotics and Autom.*, RA-3, 1987, 194–206.
3. "Geometric Ambiguities in Boundary Representations," *Computer Aided Design 19*, 1987, 141–147.
4. "Projective Blending Surfaces," *Artificial Intelligence 37*, 1988, 357–376.
5. "Algebraic Curves," in *Mathematical Aspects of Scientific Software*, J. Rice, ed., IMA Volumes in Math. and Appl., Springer Verlag, 1988, 101 – 122.
6. "Towards Implementing Robust Geometric Computations," Proc. Conf. Comp. Geometry, Urbana, Ill., 1988.
7. "Tracing Surface Intersections," *Computer Aided Geometric Design 5*, 1988, 285–307.
8. "Model Generation and Modification for Dynamic Systems from Geometric Data," Springer NATO ASI Series F-50, 1988, 481–492.
9. "The Problem of Accuracy and Robustness in Geometric Computation," *IEEE Computer 22*, 31–42.
10. "Local Implicitizations of Curves and Surfaces," *ACM Trans. on Graphics*, to appear 1989.
11. "Robust Boolean Operations on Polyhedral Solids," TR-87-875.
12. "A Dimensionality Paradigm for Surface Interrogation," TR 88-837; submitted to CAGD.
13. "On the Geometry of Dupin's Cyclide," *The Visual Computer 5*, to appear in June.
14. "Variable Radius Blending with Dupin Cyclides," to appear late 1989.

6.3 Workshops Organized

1. "Computational Issues in Robotics," IMA Minnesota, August 1987.
2. "Blending Surfaces," Minisymposium, SIAM Conf. on Applied Geometry, Albany 1987.
3. "Algorithmic Aspects of Geometry and Algebra," MSI Cornell, July 1988.
4. "Applying Algebraic Geometry to Surface Intersection," Short course, SIGGRAPH 88.
5. "Computing about Physical Objects," Minisymposium, SIAM Conf. on Applied Geometry, Arizona, November 1989.
6. "The Computational Paradigm In Science and Engineering," Symposium at the annual meeting of the American Assoc. for the Advancement of Science, New Orleans, February 1990.

6.4 Invitations to Workshops

1. NSF Workshop on Geometric Reasoning, Oxford, July 1986.
2. IMA Workshop on Supercomputing, Minnesota, March 1987.
3. NSF Res. Conf. Geometric Modeling and Robotics, Detroit, May 1987.
4. Summer Program on Robotics, IMA Minnesota, August 1987.
5. NATO Workshop on CAD Based Programming for Sensor Based Robots, Italy, July 1988.
6. Trento School on VLSI Design and Parallel Algorithms, Italy, July 1988.
7. MSI Workshop on Gröbner Bases, Cornell, 1988
8. NSF-IFIP Workshop on Solid Modeling, Rensselaer, September 1988
9. Oberwolfach Workshop on Applicable Algebra, West Germany, January 1989.
10. NSF Workshop on Information Technology, Atlanta, March 1989
11. Oberwolfach Workshop on Surfaces in CAGD, West Germany, April 1989
12. NATO School on CAGD, Canary Islands, July 1989.

6.5 Talks at Universities and Labs

1. General Electric, Schenectady, 1987
2. Courant Institute, New York, 1987
3. Carnegie-Mellon University, Pittsburgh, 1988
4. USC, Los Angeles, 1988
5. University of Washington, 1988
6. University of Maryland, 1988
7. University of Linz, Austria, 1988

6.6 Editorial Responsibilities

1. Editor, SIAM Frontiers Series on Applied Geometry.
2. Editorial Board, Journal of Symbolic Computation.
3. Editorial Board, Journal for Applicable Algebra.
4. Editorial Board, Computer-Aided Geometric Design.

6.7 Professional Duties

1. Panel, NASA-CESDIS Grants, 1988
2. Panel, NSF CISE-SS Infrastructure Grants, 1988
3. Site review, NSF CER program, Univ. Rochester, 1988
4. Program committee, ACM Symp. Computational Geometry, 1989

6.8 Software and Tools

1. Box and rectangle intersection algorithm. Used to speed up polyhedral intersection algorithm.
2. Dual mode polygonal intersection algorithm. Test case for the dual mode polyhedral intersection algorithm.
3. 3D surface intersection algorithm. Explore the capabilities and limitations of purely numerical approaches.
4. Planar curve tracing algorithm using desingularization. Proof of concept: Numerical and symbolic computation can be successfully combined.
5. Interface between S-geometry and Karasick's polyhedral modeler. Tool to study user interfaces.
6. Newton system. Proof of concept: Automatic model construction, modification, and analysis from geometric data is possible.
7. Linear equation solver for distributed computation. Part of an effort to construct a distributed version of the Newton system.
8. Interface between Newton system a[illegible]EARN, a structural mechanics finite element package. Proof of concept: Finte element techniques can be interfaced with the Newton system.
9. Dual mode polyhedral intersection algorithm (in progress). Test bed to study robustness issues.
10. Device independent display algorithm for algebraic surfaces. Visualization tool for surface research.
11. Surface intersection algorithm in arbitrary dimensions. Proof of concept: Higher dimensional problem formulations can be used directly, and have important practical benefits.
12. Elimination algorithm. Proof of concept: Verify that we can speed up expensive symbolic algorithms by reducing their generality.
13. Point projection onto curves and surfaces (in progress). Proof of concept: Assess practical usefulness of differential geometry in modeling.

14. **Track projection of a curve onto a surface (in progress). Proof of concept: Assess practical usefulness of differential geometry in modeling.**

15. **Surface polygonalizer, implicit or parametric surfaces. Used for visualization and mesh generation.**

Most of this software has been developed in Common Lisp for Symbolics Lisp machines, with the exception of the display algorithm and the higher dimensional surface intersection algorithms which are written in C for Unix machines.

7 General Research Directions

The following themes are considered to be of critical importance to promoting the utility of computation in science and engineering, especially in manufacturing.

1. **Research into the substrata problem in geometric modeling. What can we do to design correct implementable algorithms with the needed performance? Can we retro-fit methods onto existing modelers that increase robustness? Can we devise "approximate" models, either in the sense of tolerance, or in the sense of statistical variation?**

2. **Integrate symbolic and numerical methods. *Very few examples* can be cited in which the best aspects of each approach have been combined. There should be many trade-offs, but what we know is only anecdotal.**

3. **Develop conceptual geometric design. What is a feature? What is design detail, what is conceptual design? At this time, even case studies would be useful. Case studies might consider specific applications in aircraft wing design, ship hull design, and space applications.**

4. **Develop conceptual functional design. What is the interaction between feature and functionality? How do tolerances affect functionality?**

8 Hilbert-Size Problems?

1. **Given a 3-dimensional geometric object, remove all surface structures of size smaller than a given tolerance ϵ.**

2. **Given an algebraic equation $f = 0$ in the variables $x_1, ..., x_m$ and of degree n. Find k algebraic equations $h_1 = 0, ..., h_k = 0$, in $m + r$ variables, such that the algebraic degree of each h_i is strictly less than n and the projection of the algebraic set defined by the h_i, onto the subspace defined by the $x_1, ..., x_m$, is the algebraic set of f.[1]**

3. **Given n rigid objects in contact, each acted upon by known external torques and forces; devise an efficient algorithm to determine all contact forces.**

[1] **For example, given a quartic curve $f(x, y) = 0$, can it be obtained as projection of the intersection of two quadrics, $h_1(x, y, z) = 0$ and $h_2(x, y, z) = 0$?**

4. **Give a definition of feature and show that it is unambiguous. Then devise a recognition algorithm.**

A COMPUTATIONAL LOGIC AND AUTOMATED REASONING:

Report for Idaho ONR Computer Science Workshop

Bob Boyer
Matt Kaufmann
J Moore

Computational Logic, Inc.
June 1989

0. Abstract

The development of software engineering as a discipline has been influenced substantially by the development of formal, mathematical techniques for reasoning about computer programs. One of the most promising avenues of research is to develop formal mathematical theories for specifying and proving the correctness of computer systems. The key idea here is that if one produces a proof that a computing system satisfies its specification, then the only reasons for which the system can fail to work as intended are (a) hardware failure and (b) the failure of the formal specifications to capture intended behavior. Because the development of formal proofs is itself a very error-prone and tedious activity, we have been pursuing, partially with ONR support, the development of a computer program for checking proofs of correctness of computer systems. We have made substantial progress in this direction using constructive mathematical theories, and with ONR support we are continuing to extend our work through mathematical theories of greater power.

1. Background.

Boyer and Moore began collaboration on their mechanized logic and theorem prover in the early 1970s. A summary of their work through 1979 is given in [ACL79]. The following excerpt is taken from [ACLH88] and describes some of the research stimuli for Boyer and Moore during the last few years.

> .... perhaps the most important change since the publication of "A Computational Logic" was that in 1981 we moved from SRI International, where we were involved exclusively in research, to the University of Texas at Austin. Our research home at the University of Texas was the Institute for Computing Science. However, as professors in the Department of Computer Sciences, we teach. In 1981 we began teaching a graduate course, now called "Recursion and Induction", on how to prove theorems in our logic, and we initiated a weekly logic seminar attended by graduate students, other faculty members, and logicians from local research organizations. These efforts dramatically increased the number of people familiar with our work. In addition, we began using the theorem prover to check the proofs of theorems we wanted to present in class (e.g., the unsolvability of the halting problem).

Kaufmann first became involved in this work in the context of adapting

the Boyer-Moore prover to functional language verification while at the Burroughs Austin Research Center, 1984-86. He joined the Institute for Computing Science at the University of Texas (where Boyer and Moore were located) when that Center was closed down in the summer of 1986. His previous experience as a mathematical logician has helped stimulate the current push to add capabilities in first-order quantification and set theory.

2. Research Objectives.

The long range objectives of the research include

** enabling programmers to produce software that is mathematically proven to meet its specifications by using mechanical theorem-proving programs that check proofs

** supporting proofs of correctness of computing systems to provide a trusted base for those applications

3. Research Issues.

The key idea here is that if there is a proof that a computing system satisfies its specification, then the only reasons for which the system can fail to work as intended are (a) hardware failure and (b) the failure of the formal specifications to capture intended behavior.

4. Approach.

The prover was developed as program verification system partially under ONR support. A basic version of the logic and prover is documented in [ACL79]. Enhancements to the logic and prover are built on that base in our approach. A more up-to-date version of the logic and prover, documented in [ACLH88], illustrates this approach by documenting the following extensions to the prover and logic.

** a hints facility which allows a significant measure of user control over the prover

** a fully integrated decision procedure for linear arithmetic

** a facility for "metatheoretic extensibility", i.e. for allowing the user to extend the theorem prover in a provably sound manner

** NQTHM: an extension to support partial functions, bounded quantification, and an interpreter for the logic within the logic

Three principles are fundamental to our approach.

** The logic is completely specified and the prover is implemented with extreme care so that it soundly implements the logic.

** The criterion for success of the prover is its successful application to specific theorems to prove.

** Prover use is an important part of the process of deciding how to extend its capabilities.

Let us note that many other formal methods exist which have varying degrees of mechanical proof support. Some of these, such as the Edinburgh Logical Framework, are concerned more with foundations than with applications to program verification. Others such as Z and VDM (which are gaining popularity especially in Europe) emphasize formal reasoning without the benefit (or burden!) of mechanical proof support. Among those systems which emphasize mechanical proof support, ours is unsurpassed in the collection of theorems which have been proved (or "proof-checked") using the system. The following section shows that the logic and its proof support are very live research areas and we feel quite strongly that our most productive research route is to build heavily on our previous work. This may involve some rather serious changes; for example, the V&C$ and EVAL$ approach to bounded quantification may be replaced by different additions to the logic. However, we believe that our basic approach of using recursion and induction with a formal constructive logic is a sound one on which we will continue to build. We will continue to get feedback from users of the system as a means for improving its utility.

5. Payoffs

The ultimate payoff of this technology is the ability to mechanically proof-check mathematical properties, especially of computer software and hardware. Many individuals have successfully mechanically proof-checked theorems in the following areas: elementary list processing, number theory, metamathematics, bounded quantification, communication protocols, concurrent algorithms, Fortran programs, real time control, assembly language implementation, operating system implementation, compiler correctness, hardware verification, and set theory. Several of these efforts have become the main components of doctoral dissertations.

Another kind of payoff is the progress that our approach has had in improving the mechanical tools for the logic and even the logic itself. Such progress includes:

** an interactive proof-checker enhancement
** an improved facility for reusable theories
** an extension to the logic and prover to allow "partial definitions" and functional instantiation
** preliminary extensions to the logic, prover, and proof-checker to allow first-order quantifiers and a rudimentary capability in set theory
** an experimental extension to improve reasoning power for equivalence relations
** an experimental extension to the execution environment to allow constant-time array access and update while remaining in a purely functional framework

6. Research Directions

The following subsections lay out some of our current research directions in automated reasoning and program verification. We also are involved in a number of applications of this technology. For example, one of our primary focuses at Computational Logic, Inc., is on "trusted systems", i.e. on provably correct implementations of high-level languages on hardware. Our position paper for the upcoming ONR-sponsored "Workshop on Directions in Software Analysis and Testing

[ONRWorkshop] gives an overview of this line of research. And there is also research underway in the application of the Boyer-Moore logic and its proof support to the mechanical verification of properties of distributed progr ms and floating-point algorithms. However, we confine ourselves below to those areas related to extending our cabilities in automated reasoning and program verification.

a. First-order quantification and set theory

Reasoning about computer systems requires skill in two distinct types of mathematical theories: the constructive and the set theoretic. On one hand, arguments must be made about the elementary constructible objects that one actually finds in computers, such as integers, finite lists, and strings. On the other hand, in order to specify the interaction of computing systems with the real world and to specify the interconnection and interdependence of computing systems, one needs the full range of mathematical concepts, such as are usually developed within set theory. For example, both of the systems currently approved by the DoD for A1 security level certification, Gypsy and FDM, utilize set theory.

Under support from ONR, we have developed a program specification and verification system which is unsurpassed in its facility for making inferences within a constructive theory. So far, however, no program verification system has been developed in set theory with comparable power. Recent progress made under ONR support suggest a method for adding set theory and related mathematical concepts to our system. The main idea is to add an interface from first-order logic to the Boyer-Moore logic, and to generalize the notion of Skolemization to extend this interface to expressions that contain set-builder notion. This research involves both (a) theoretical research in the selection and formulation of a precise set theory and quantification theory, (b) practical research in the implementation of a theorem-prover capable of making automatic inferences about questions in the selected theory, and (c) demonstrations of the applicability of the developed theorem-proving techniques (ultimately, to the verification of substantial computing systems).

We do not currently intend to add first-order quantifiers and set theory as primitives in the logic. Such a radical decision would probably require wholesale recoding of the theorem prover, for example because of bound variables, and possibly some wholesale reworking of its heuristics, which currently are based largely on the recursion-induction duality and rewriting but not unification. Instead, we are pursuing an approach which uses Skolemization as an interface from first-order logic to the constructive logic currently in use. We have already enjoyed preliminary success in this venture by proof-checking formalizations of Cantor's theorem that the power set of a set is not of the same cardinality of the set, of Koenig's tree lemma, of the infinite exponent-2 Ramsey Theorem, and of the Schroeder-Bernstein Theorem.

In order to extend the set-theoretic capabilities of the system, our initial approach will be to introduce sets as objects. However, we will provide "set-builder notation" only as syntactic sugar. Specifically, we believe that we can successfully extend Skolemization from first-order logic into the realm of set theory by using it to eliminate set-builder expressions.

We will extend and use the interactive proof-checker, PC-NQTHM, to begin to look for useful proof methods and heuristics in the extended logic. Most of the extensions to PC-NQTHM will be in the form of macro commands that expand into sequences of primitive commands; this macro facility is already in heavy use and gives us the ability to add new functionality to the proof checker without risking soundness. Extensions to the logic are most easily accomplished in PC-NQTHM where it is unnecessary for us to implement heuristic controls on the applications of new rules of inference or axioms. Once PC-NQTHM is extended we will begin to use it to prove many theorems in set theory and related applications. As we develop and codify the heuristics for manipulating the new concepts we will add new proof heuristics to NQTHM.

We will probably investigate these issues a bit more before bringing in a graduate student, in order to provide reasonable guidance. If we do not find a graduate student interested in pursuing this area for dissertation research, we will probably hire a couple of students to exercise a system that we build.

Note that we do not in general encourage students to build new general-purpose automated reasoning systems. The Boyer-Moore prover is the product of some 30+ man-years of effort; hence we expect it to continue to be rare that someone with limited experience can build a state-of-the-art general-purpose theorem-prover. However, we do believe that the pursuit of provers with selected strengths is a reasonable research topic for a graduate student. In particular, as we discussed above, we expect to support a student under our current ONR contract to pursue research in the directions outlined in the paragraphs above.

b. Other extensions of the prover

We plan to extend NQTHM towards an open-architecture formal reasoning system. This will include implementing an advanced library mechanism, continuing to develop and disseminate reusable theory libraries, implementing equivalence reasoning, increasing support for team proof development, extending the interactive proof checker, and studying integration with the Argonne prover, OTTER.

We also plan to extend NQTHM's heuristics to include congruence-based rewriting. This can be seen as a step toward an open-architecture system: the current NQTHM has many special-purpose heuristics for one built-in primitive ("=") that can be made more generally available for user-defined relations at the cost of formalizing the interface (namely, the idea of congruence relations). We will also look for other ways in which NQTHM's special purpose heuristics can be "opened up" so that users can get the power of those heuristics applied to more general concepts.

c. Lisp verification

One recent exciting area has been the application of our system to the proofs of properties of Lisp programs. More precisely, we have built a system for reasoning about programs written in a language which satisfies the language definition requirements for a subset of Common Lisp. We call this language "Rose Common Lisp" or "RCL". The existing prototype system (see [RCL]) is actually a translator from Common Lisp definitions to definitions in the logic supported by

NQTHM; because of the Lisp-like nature of NQTHM's logic and the care taken in defining the translator, many RCL programs translate to nearly identical NQTHM functions. However, the system handles non-applicative constructs such as: assignment, both for local (LET) and global (special) variables; explicit flow of control, both with local and non-local exits (CATCH and THROW, BLOCK and RETURN-FROM) and with "go-to" (PROG); property lists; and macro definition.

The significance of this effort lies largely with the fact that the language in question is an implementation of Common Lisp. To the best of our knowledge there do not exist verification systems for any "real languages" which have even the power of the existing prototype. By "real languages" here we mean ones that are dialects of languages in everyday use by programmers who have no special interest in formal verification.

We will work many more small examples in the course of further developing the system. However, we propose to demonstrate the feasibility of our approach by verifying a significant application, such as a portion of NQTHM (e.g., the "type-set" mechanism which determines the type of an expression, the "clausify" mechanism that converts an IF-expression into clausal form, the pattern matcher that finds instances of rewrite rules).

To support technology transfer, RCL will ultimately be written in RCL, in a manner such that we expect it to run correctly on any Common Lisp implementation. It will therefore be highly portable. In additior, we will carefully document the final system so that it is accessible to Lisp programmers outside of Computational Logic, Inc.

7. Grand challenge.

OK, how's this?

Prove the correctness of the implementation of a functional programming language with respect to its denotational semantics.

Or this?

Formalize the real numbers using set theory (by way of Dedekind cuts, say), and prove some properties of the reals as well as some properties of some simple algorithms over the reals.

Actually, a more close-to-home goal is to extend the CLI "verified stack" work [ONRWorkshop] to provably correct running execution environments, encompassing the high-level language level down to the register-transfer hardware level.

8. Research transitions.

A near-term beneficiary of this research is anyone who wishes to formally, and with assurance, prove mathematical properties. In particular such properties might be correctness of software and hardware systems, and in that sense we are already our own customer at Computational Logic, Inc., with the various "trusted systems" proofs. But who outside our group might be interested in mechanical verification of mathematical properties?

The research community is of course one obvious place to look for

those who might take advantage of our work. Some of the obvious ways we will continue to transfer this technology to that community are by way of books, technical reports, journal publications, and conferences. In addition, all three of us teach some courses at the University of Texas at Austin, and of course this spreads the technology into the university community. (Of particular relevance to our current ONR contract is the fact that Kaufmann will be teaching a graduate-level Set Theory course this fall.)

The Rose Common Lisp project described above is one method we see for bringing this research into practical use by software engineers whose primary interest is not program verification or automated reasoning. With that project we envision opening up this technology to the general community of Lisp programmers.

Our group is also investigating collaboration with hardware manufacturers in using our methods to help produce correct hardware.

The Boyer-Moore theorem prover is actually a highly interactive tool. Nevertheless, some users have found that the enhanced interactive capabilities offered by the PC-NQTHM proof-checker, which was developed primarily under ONR support, provide a useful interface to the logic. We will continue to maintain PC-NQTHM to keep it up-to-date with the latest version of NQTHM.

Program verification is still a primary intended use of the results of our research. However, in spite of the practices outlined above, there is still a large gap between what we do on a daily basis with our tools at Computational Logic, Inc., and what the rest of the world is doing. We need to look for more ways to bring our work into the mainstream. An example of a rather recent move in that direction is the addition of the book mechanism to the prover, which will facilitate reuse of theories and team collaboration in much the same way that modern software development environments are supposed to encourage reuse of code and programmer cooperation. But we need to keep looking for more ways to bring our technology into the mainstream of the software development process.

9. Technological impacts.

Everyone likes more computing power. Nevertheless, at this time we are reasonably content with our equipment. We do anticipate a use for parallel architectures, especially for the rapid replay of proof files. In fact we have recently constructed, with ONR support, such a capability on a network of Unix machines.

10. Societal Issues and Miscellaneous Flaming.

We're happy that our contract is multi-year. Proposal writing is extremely time-consuming.

11. Recommendations to Funding Agencies

<<none>>

REFERENCES

[ONRWorkshop] "Verified Program Support Environments," Internal Note 143, Computational Logic, Inc., June 1989.

[RCL] M. Kaufmann, "A Verification System for a Subset of Common Lisp", Internal Note 110, Computational Logic, Inc., January, 1989.

[ACLH88] R. S. Boyer and J S. Moore, "A Computational Logic Handbook", Academic Press, Boston, 1988.

[ACL79] R. S. Boyer and J S. Moore, "A Computational Logic", Academic Press, New York, 1979.

Incremental Computation

Tim Teitelbaum
Department of Computer Science
Cornell University
Ithaca, NY 14853

June 8, 1989

1 Background

Our research for more than ten years has focused on environments for editing complex structured objects: computer programs, proofs of theorems, program specifications, spreadsheets, and the like. Our premise has been that integrated systems that provide immediate feedback during the creation and transformation of these objects provide substantial improvements in productivity over traditional "batch" systems. The most frequently cited example of this type of system is a programming environment that tightly couples tools for program editing, browsing, analysis, transformation, execution and debugging.

Our early work in this area culminated in the development of the Cornell Program Synthesizer [TR81], a highly integrated environment for a small subset of PL/I. The Synthesizer graphically demonstrated the feasibility of building a self-contained, highly interactive environment that supplanted many of the traditional batch-oriented development tools. Whereas it was far too limited to serve as a tool for professional programmers, it was used successfully as a tool for teaching top-down structured programming. In the space of several years it served more than 20,000 introductory programming students at Cornell and other universities.

It quickly became apparent that "hard-coding" an environment for a specific language, as was done with the Synthesizer, was the wrong approach. The most challenging aspect of building Synthesizer-like systems is the problem of efficiently maintaining derived context-sensitive information as the underlying object changes; for example, updating object code

after each editing modification to source code. We believed that this problem was amenable to a generic solution and accordingly embarked on a study of incremental computation.

2 Research Objectives

Our long term objective has been to develop a comprehensive theory of incremental computation that allows cost-efective re-use of previous executions. We maintain that incremental computation has the potential to reduce processing time significantly for a wide variety of applications.

Our interest in incremental computation stems from a narrower and more immediate objective of improving productivity by showing how to design and implement effective programming environments and environments for formal reasoning. We maintain that such environments can make excellent use of the methods we develop for incremental computation.

3 Research Issues

The problem of incremental computation can be posed in the following terms. Let $\mathcal{F}$ be a computable function mapping $X \rightarrow Y$, two arbitrary domains. Let $x_0, x_1, ..., x_n$ be a sequence of values in X, where the distances between successive values of x are small, for some notion of distance in X. We wish to compute $\mathcal{F}(x_0), \mathcal{F}(x_1), ..., \mathcal{F}(x_n)$ in an on-line fashion; i.e., each $\mathcal{F}(x_i)$ must be computed without reference to subsequent values $x_{i+1}, ..., x_n$. We say that $\mathcal{F}$ is computed *incrementally* if each computation of $\mathcal{F}(x_i)$ takes advantage of the fact that we have already computed $\mathcal{F}(x_0), ..., \mathcal{F}(x_{i-1})$.

By definition, then, incremental computation is a subject with broad applicability. Since $\mathcal{F}$, X, and Y are arbitrary, we see incremental evaluation as a research area whose domain spans a wide spectrum of computable problems. $\mathcal{F}$ might be a function that inverts a matrix x, finds the transitive closure of graph x, compiles a high-level language program x, loads a set of object modules x, or computes navigational information for an aircraft from sensor data x.

The significance of incremental computation lies in its potential to dramatically reduce processing time across this broad problem domain. In non-incremental computations, each evaluation of $\mathcal{F}(x_i)$ is ignorant of the intermediate results of any previous execution. Yet,

in many applications, when x changes just slightly, the bulk of the computational steps involved in computing $\mathcal{F}(x_{i+1})$ remain the same. For example, recomputing a matrix operation after the change of a single cell can often reuse intermediate results from a previous computation. Similarly, the object code produced by a compiler after a single line change in a program differs little from that before the change. Incremental computation can offer a dramatic savings in machine cycles by re-using unchanged intermediate results from previous computations of $\mathcal{F}$ and only evaluating the subset of the problem that is affected by the change from x_i to x_{i+1}. By bypassing complete reevaluation of previous intermediate results, incremental algorithms can have asymptotically better running time than the non-incremental alternative.

We believe that the general application of incrementality will accelerate the trend that has shortened computer response time to changing inputs. Just as improved hardware and systems software precipitated the transition from batch to interactive systems, incremental evaluation allows what we call *immediate systems*. This computational model resembles the well-known spreadsheet, in which rapid recomputation permits a "what-if" approach to problem solving. By experimenting with various inputs to a problem, the user of the spreadsheet can arrive at an optimal solution. Incremental computation applied broadly can extend this spreadsheet-like interaction to complex problems like interactive theorem proving systems and programming environments where program results are immediately reevaluated as input and source are edited.

Although immediate systems are the most natural application of incremental evaluation, this technique is also effective in systems where a user is not directly involved in the computational process.

In real-time applications, for example, quick response to asynchronous changes in individual sensor values is required. Viewing the collection of sensor data as a vector input quantity x and the required setting of control parameters as the result of a vector valued function $\mathcal{F}$, such a program must respond to each change in a component (or subset of the components) of the input x. Incremental computation of $\mathcal{F}$ in response to small changes in sensor data may be required in order to meet real-time requirements.

Any program that alternates data analysis with data transformation may be a candidate for incremental evaluation. Consider, for example, the code optimization phase of a compiler. Typically, extensive data flow information is gathered by analysis of the program control flow graph. Based on this information, the compiler selects a code-improving transformation, which produces some modification of the control flow graph. In general, the modification of the control flow graph invalidates the derived data flow information, which must then be updated before the next transformation can be selected and applied.

Using the notation described earlier, $\mathcal{F}$ is the function that computes data flow information from a control flow graph argument. Each optimizing program transformation represents an incremental change in the control flow graph. Repetitive application of $\mathcal{F}$ to slightly different control flow graphs suggests that the incremental computation of $\mathcal{F}$ could improve the performance of the code optimization phase.

4 Approaches

Much research into incremental computation has taken an *ad hoc* or *algorithm-dependent* approach. In this approach, an existing algorithmic solution to a specific problem, $\mathcal{A}$, is modified to produce a new algorithm, $\mathcal{A}'$, that computes a result in response to changes in x. For example, [RM87] and [Zad84] both describe incremental approaches to the data flow analysis problem described above. The advantage of restricting the domain of the problem to specific algorithms is that each problem may offer its own unique opportunities for incrementality. There is considerable research yet to be done on ad hoc incremental algorithms, including finding good incremental methods for classical algorithms, developing general paradigms for deriving good incremental algorithms, and formulating classifications that characterize degrees to which algorithms can and cannot be made incremental.

The focus of our research, however, is the more general, *algorithm-independent* approach to incrementality. In this approach, the problem of incrementality is orthogonal to the design of an algorithm $\mathcal{A}$ for a specified problem. Ideally, $\mathcal{A}$ can be designed and programmed in a standard, non-incremental manner. These algorithms are often simpler to design and implement than those that are explicitly incremental, producing programs whose correctness is easier to verify. The translation from the non-incremental algorithm $\mathcal{A}$ and implementation $\mathcal{T}$ to the incremental implementation $\mathcal{T}'$ is automated. The research problems in this area include the creation of programming languages in which evaluation of $\mathcal{T}$ can be efficiently updated when the input x changes, identification of abstract data types of general utility for which efficient incremental updating algorithms can be devised, and the classification of problem domains for which the algorithm-independent approach is viable.

Approaches to incremental computation have incorporated two distinct principles: taking advantage of the *history* of previous computations in computing new results, and determining the effect on the output of a relatively small *change* to an input. These two distinct approaches can be illustrated by the work of other researchers in the ONR-supported computer science community:

- The persistant data structures of Driscoll, Sarnak, Sleator, and Tarjan [DSST89] address a key issue in the use of histories for incremental computation: how the multiple versions of a data structure that arise in the course of the computation can be maintained in a space-efficient manner that still permits time-efficient access to any version.
- The finite differencing of Paige [PK82], a generalization of optimization by strength reduction, replaces expensive local calculations made inside loops with incremental counterparts that make only small changes to large non-local data structures. An earlier paper of Earley ([Ear76]), on which Paige's work is partially based, pointed out that such optimization techniques could prove equally beneficial in implementing incremental algorithms, as opposed to using them only to improve non-incremental ones.

Our contributions to incremental computation have involved both the use of histories and the propagation of small changes.

5 Progress

In the Synthesizer, semantic analysis had been expressed imperatively, requiring every semantic action to have a corresponding *undo* action. In our new approach, we identified [DRT81] attribute grammars [Knu68] as a promising alternative. Their descriptive power makes attribute grammars applicable to a wide variety of objects and their declarative nature eliminates the need for explicit "undo" actions. In this framework, complex objects are represented as consistently attributed derivation trees with respect to a given attribute grammar. Whenever the object is modified, attribute values are updated to restore the consistent state defined by the attribute equations. We developed the theoretical foundations of this approach to building incremental systems in a series of papers that culminated in Reps' Ph.D. Thesis, recipient of the 1983 ACM Doctoral Dissertation Award [Rep84]. The contributions of this work were as follows:

- It proposed the attribute grammar model of incremental computation and argued its advantages.
- It contained optimal evaluation algorithms, not just for arbitrary noncircular attribute grammars, but for the absolutely noncircular and the ordered attribute grammar subclasses as well.

- It presented two algorithms that carry out attribute evaluation while reducing the number of intermediate attribute values retained. While others had worked on this problem, these algorithms were the first to achieve sublinear worst-case behavior.
- It emphasized the importance of environment generation as opposed to ad hoc construction techniques. In this, we were certainly not alone. Rather, this message reenforced that of Emily, Mentor, and Gandalf.
- It demonstrated the possibility of applying formal techniques and rigorous analysis to a fundamental software engineering issue and stimulated others to work on the problem of incremental static-semantic analysis.

Our early work made several simplifying assumptions that are not valid in practice. First, our notion of optimality charged for changed copy attributes, whose only function is to communicate a value from one point in the tree to another. Second, our notion of optimality charged for *all* uses of a changed aggregate-valued attribute, even when only a single component of the aggregate changes. Third, we assumed that each semantic function is a constant-time operation. These shortcomings were partially addressed in Hoover's Ph.D. Thesis [Hoo87], whose main contributions were as follows:

- It introduced a data structure called the structure tree, which allows transitive dependencies to be easily and efficiently represented in the attribute dependency graph and maintained in the presence of changes to the dependency graph.
- It showed how structure trees can be used to improve incremental evaluation performance when the dependency graph contains copy rule chains and aggregate-valued attributes.
- It presented a new, heuristic incremental evaluation algorithm that appears to works well in practice, although its running time is not guaranteed to be linear in the number of attributes changed. This new algorithm was required by the introduction of nonlocal structure tree edges, i.e. transitive dependency edges not properly embedded in the derivation tree.

The attribute grammar approach to incremental computation is primarily history-based — the collection of saved attributes are essentially a history of the intermediate values that arise in the course of a computation. Incremental attribute evaluators work by restarting the computation in the middle, at exactly the right place with respect to the object's mutation. However, Hoover's solution to the problem of aggregate attributes introduces elements of

the alternate approach to incremental computation, the propagation of small changes, and bears a resemblance to Page's finite differencing technique.

Attribute grammars are only suitable for some problems. For instance, if attention is restricted to noncircular attribute grammars and unit-time semantic functions, only linear-time algorithms can be expressed. To circumvent this limitation, attribute grammar systems typically allow the use of arbitrary recursive semantic functions — for which incremental attribute evaluation schemes provide no incrementality. Motivated by this observation, we have begun an investigation into the use of function caching, or memoising, to provide incremental computation within function evaluation [Pug88b] [PT89]. Function caching may be used in a hybrid system in conjunction with incremental attribute evaluation, or possibly, may emerge as a complete alternative to the attribute grammar approach to incremental computation.

The implementation of the Synthesizer Generator [RT88a] [RT88c], a tool for creating Synthesizer-like language-based environments from formal specifications, has given us the opportunity to both prototype these research developments and develop a platform for future research work. The research uses of the Synthesizer Generator within our group have included the following:

- The original T. Reps attribute evaluation algorithm [Rep82] [Rep83] has been part of the Synthesizer Generator since its first release.
- The incremental attribute evaluator developed by T. Teitelbaum and T. Reps [RT88b] is currently the most efficient evaluator in the released system for grammars that fall into the class of *ordered attribute grammars* [Kas80].
- S. Horwitz used the Generator to study and prototype a specification formalism based on a coupling of attribute grammars and relational databases [HT85] [Hor85].
- R. Hoover's work on incremental graph evaluation [Hoo87] [Hoo86] [HT86] is implemented in the latest version of the Generator. This new attribute propagation algorithm, in many cases, significantly reduces the size of the set of attributes that must be reevaluated after a tree modification.
- W. Pugh's work on general models of incremental computation using lazy structure sharing and memoizing [PT89] [Pug88a] [Pug88b] was prototyped in the Generator.
- S. Peckham is currently using the Generator to examine extensions of attribute propagation algorithms that efficiently handle multiple, asynchronous modifications [Pec88]. This continues work done earlier in our research group [RMT86].

Among our users, we are aware of the following publications in which experience with the Generator has been reported: [GP] [KKM87] [vE89] [NHWG88] [NL88] [FZ] [FZCL88] [NBK88] [CR87] [Bru87] [Slo87] [BKJ88] [CP89] [BV87] [Gri87].

6 Research Directions

Storing input/output pairs for a function in a cache makes it possible to avoid repeated re-calculation of the function on *exactly the same input*. However, this technique only partially addresses the problem of avoiding redundant calculations on composite objects that may be only *slightly* altered. More problematic still is the handling of changes in *functional* values, which occur quite naturally when specifying the semantics of programs or other complex objects or systems in a denotational style. We believe that both of these problems can be addressed using the lambda-calculus, a formalism in which both functions and composite objects can be represented.

The problem of incremental re-evaluation of lambda-terms can be expressed as follows. Given a lambda-term M which reduces to normal form N, alter M slightly to yield M' (this may correspond to editing a functional value or some composite object). We then wish to determine N', the normal form of M', using as much of the information already known from the reduction of M to N as possible.

The set of intermediate lambda terms produced in the reduction of M to N provides a history of N's computation. Our idea is to examine N's history to determine exactly what parts of the computation depend on the changed part of M, and what parts do not. Then, in principle, only those subcomputations depending on the changes to M must be recomputed. Other subcomputations are invariant and can be incorporated into the new result.

An optimal incremental evaluation of M' is one that repeats no reduction already performed in the evaluation of M. Our research has focussed on formal techniques for analyzing reductions to determine dependencies on the initial term, determining exactly which intermediate values need to be remembered to enable incremental re-evaluation and which can be safely discarded, and practical means for performing reduction such that these values can be computed and stored automatically.

The problem of incremental reduction of terms in the lambda-calculus has also lead us to the study of new schemes for evaluation of lambda terms. It turns out that all existing lambda-calculus evaluation techniques are sub-optimal, in that they perform some redundant or unnecessary calculations. In most practical settings (i.e., execution of func-

tional languages), the conventions used by programmers and commonly used compilation techniques minimize the impact of such extra work by the evaluator. However, in a setting where functions are edited and interpreted on the fly, (as in practical implementations of denotational semantics), the drawbacks of existing interpreters for the lambda calculus become more apparent.

7 Grand Challenge

Discover how to create incremental software from non-incremental software automatically.

8 Research Transitions

The wide acceptance of the Synthesizer Generator as a research tool by the domestic and international Computer Science community has been one of the most satisfying results of our work. The fact that over 250 sites have licensed the Generator since its initial release indicates its value to the computer science community. The licensees are approximately one-half domestic and one-half overseas; half are within universities and colleges and half in industrial and government settings. The growth in the number of sites licensing the Generator has been essentially linear since the first release. Roughly half of the 150 receiving Release 1 ordered Release 2. This suggests that while perhaps half of our over 250 sites are mainly curious, the remaining sites are making serious use of the system.

The implementation of the Synthesizer Generator has been a side-effect of our primary work, fundamental research in incremental computation. Our implementation effort, therefore, has largely focused on "proof of concept" rather than concentrating on the overall applicability of the system. For example, although the Synthesizer Generator has been a testbed for the design of asymptotically efficient algorithms (important because they scale up), little effort has been devoted to the system's raw performance. Thus, while there is little doubt of the essential technical feasibility of our basic research ideas, much engineering remains to be done before their potential is fully realized. Such an effort is better undertaken in a commercial rather than an academic setting. The mission of the newly founded firm of GrammaTech, Inc. is to establish the Synthesizer Generator as a fully-engineered commercial product.

Appart from the success of the Synthesizer Generator per se, we believe that our research ideas have been well-received and have established something of a following for the attribute

grammar approach to building incremental systems. In addition to our software, Reps' award-winning Ph.D. Thesis has been notably influential.

9 Recommendations to Funding Agencies

Consider creating an initiative in incremental computation.

References

[BKJ88] N. Botta, E.W. Karlsen, and J. Jorgensen. The PA^{nn}dA-S editor, user's guide. Technical Report S.3.1.C1-R-11.0, Propsectra Project, Dansk Datamatik Center, 1988.

[Bru87] G. Bruns. Generating an editor for a software design editor. Technical Report STP-325-87, MCC, October 1987.

[BV87] L. A. Barford and B. T. Vander Zanden. Attribute grammars in constraint-based graphic systems. Technical Report 87-838, Cornell University, 1987.

[CP89] A. Carle and L. Pollock. Modular specification of incremental program transformation systems. In *Eleventh International Conference on Software Engineering*, Pittsburgh, Pennsylvania, May 1989.

[CR87] D. Carrington and K. Robinson. A prototype program refinedment editor. Technical report, Department of Computer Science, University of New South Wales, 1987.

[DRT81] A. Demers, T. Reps, and T. Teitelbaum. Incremental evaluation for attribute grammars with application to syntax-directed editors. *Proceedings of the 8th Annual ACM Symposium on the Principles of Programming Languages*, pages 105–116, 1981.

[DSST89] J. Driscoll, N. Sarnak, D. Sleator, and R. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38(1):86–124, February 1989.

[Ear76] J. Earley. High level iterators and a method for automatically designing data structure representation. *Journal of Computer Languages*, 1:321–342, 1976.

[FZ] P. Franchi-Zannettacci. Attribute specifications for graphical interface generation. To be presented at IFIP'89 World Congress Conference, San Francisco, August, 1989.

[FZCL88] P. Franchi-Zannettacci, B. Chabrier, and V. Lextrait. Gigas: A graphical interface generator from attribute specifications. In *Software Engineering and its Applications*, Toulouse, France, December 1988.

[GP] D. Guaspari and W. Polak. Summary: The odyssey research associates ada verification project. Ada Letters: July/August 1988. Presented at Formal Methods Committee Report during March SIGAda meeting.

[Gri87] T. G. Griffin. An environment for formal systems. Technical Report 87-846, Cornell University, 1987.

[Hoo86] R. Hoover. Dynamically bypassing copy rule chains in attribute grammars. *Proceedings of the Thirteenth Annual ACM Symposium on the Principles of Programming Languages*, pages 14–25, 1986.

[Hoo87] R. Hoover. Incremental graph evaluation. Technical Report 87-836, Department of Computer Science, Cornell University, Ithaca, New York 14853, May 1987. (Ph.D. Thesis).

[Hor85] S. Horwitz. *Generating Language-Based Editors: A Relationally-Attributed Approach.* PhD thesis, Cornell University, 1985.

[HT85] S. Horwitz and T. Teitelbaum. Relations and attributes: a symbiotic basis for editing environments. In *Proceedings of the SIGPLAN 85 Symposium on Language Issues in Programming Environments*, pages 93–106, 1985.

[HT86] R. Hoover and T. Teitelbaum. Efficient incremental evaluation of aggregate values in attribute grammars. *Proceedings of the SIGPLAN 86 Symposium on Compiler Construction*, pages 39–50, 1986.

[Kas80] U. Kastens. Ordered attribute grammars. *ACTA Informatica*, 13(3):229–256, 1980.

[KKM87] G. E. Kaiser, S. M. Kaplan, and J. Micallef. Multiuser, distributed language-based environments. *IEEE Software*, 20(4):58–67, November 1987.

[Knu68] D. E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2:127–145, 1968.

[NBK88] C.B. Nielson, N. Botta, and E. Karlsen. Experiences from using the Cornell Synthesizer Generator. Technical report, Prospectra Project, Dansk Datamatik Center, 1988.

[NHWG88] M. Nielsen, K. Havelund, K.R. Wagner, and C. George. The RAISE language, method and tools. In L. Marshall R. Bloomfield and R. Jones, editors, *VDM '88, VDM - The Way Ahead*, pages 376–403. Springer-Verlag, 1988. Lecture Notes in Computer Science 328, Proceedings of 2nd VDM-Europe Symposium.

[NL88] M. Nielsen and S Lynenskjold. RAISE project overview. Technical Report 315, Dansk Datamatik Center, November 1988.

[Pec88] S. Peckham. An algorithm for maintaining tree projections in amortized $o(\log n)$ time. Cornell University, November 1988.

[PK82] R. Paige and S. Koenig. Finite differencing of computable expressions. *Transactions on Programming Languages and Systems*, 4(3):402–454, July 1982.

[PT89] W. Pugh and T. Teitelbaum. Incremental computation by function caching. In *Proceedings of the Sixteenth POPL*, 1989.

[Pug88a] W. Pugh. An improved replacement strategy for function caching. In *1988 LISP and Functional Programming Conference*, 1988.

[Pug88b] W. W. Pugh. *Incremental Computation and the Incremental Evaluation of Functional Programs*. PhD thesis, Cornell University, 1988.

[Rep82] T. Reps. Optimal-time incremental semantic analysis for syntax-directed editors. *Proceedings of the Ninth Annual ACM Symposium on the Principles of Programming Languages*, pages 169–176, 1982.

[Rep83] T. Reps. Static semantic analysis in language-based editors. *Digest of Papers Spring Compcon 83*, pages 411–414, 1983.

[Rep84] T. W. Reps. *Generating Language-Based Environments*. The MIT Press, Cambridge, Massachusetts, 1984.

[RM87] B.G. Ryder and M.D.Carroll. Incremental data flow analysis via attributes. Technical Report LCSR-TR-93, Department of Computer Science, Rugers University, June 1987.

[RMT86] T. Reps, C. Marceau, and T. Teitelbaum. Remote attribute updating for language-based editors. In *Proc. of the 9th ACM Symposium on Principles of Programming Languages*, 1986.

[RT88a] T. Reps and T. Teitelbaum. *The Synthesizer Generator: A System for Constructing Language-Based Editors*. Springer-Verlag, NY, 1988. 315 pages.

[RT88b] T. Reps and T. Teitelbaum. *The Synthesizer Generator: A System for Constructing Language-Based Editors*, chapter Chapter 12: Incremental Attribute Evaluation for Ordered Attribute Grammars, pages 246–277. Springer-Verlag, NY, 1988.

[RT88c] T. Reps and T. Teitelbaum. *The Synthesizer Generator Reference Manual*. Springer-Verlag, 3 edition, 1988. First edition, Department of Computer Science, Cornell University, Autgust, 1985; Second edition, June, 1987.

[Slo87] E.J. Slotboom. An editor for process algebra dynamic semantics. Technical report, Unversiteit Twente, 1987.

[TR81] T. Teitelbaum and T. Reps. The Cornell Program Synthesizer: a syntax-directed programming environment. *CACM*, 24:563–573, September 1981. reprinted in *Interactive Programming Environments*, Barstow, D.R., Sanderwall, E., and Shrobe, H., McGraw Hill, 1984.

[vE89] P. van Eijk. LOTOS tools based on the Cornell Synthesizer Generator. Technical Report 89-5, Universiteit Twente, 1989.

[Zad84] Frank Kenneth Zadeck. Incremental data flow analysis in a structured program editor. In *Proceedings of the ACM SIGPLAN '84 Symposium on Compiler Construction*, 1984. Also published as SIGPLAN notices Vol. 19, No. 6, June 1984.

TITLE: Theoretical Foundations of Problem-Solving Environments

PRINCIPAL INVESTIGATOR: Robert L. Constable

LOCATION: Cornell University
Dept. of Computer Science
Ithaca, NY 14853

TELEPHONE: (607) 255-7316

R&T No.: 4331699
CONTRACT No.: N0001488K0409

SCIENTIFIC OFFICER: Ralph Wachter

Technical Objectives:
Experimental research prototypes of verification systems for computer programs and hardware designs, based on relatively recent theoretical advances in mathematical logic, programming language semantics and symbolic computation, are being developed. It is expected that new theoretical results will contributed to a substantial improvement in the power and range of applicability of these systems. The objective of this research is to provide the necessary foundations to contribute to the next generation of verification systems through basic research in type theory and its relevance to programming languages and programming environments.

Approach:
The contractor will investigate issues related to the polymorphic type theory, meta-level reasoning about program properties, and partial objects in constructive type theory. In particular, a new reasoning principle, known as *computational induction*, has been discovered. It is founded on the concept that if a partial function halts on some input, then the computation of that function applied to that value is in some sense well-founded. The computational induction rule is a important theoretical contribution to type theory, but its formulation raises very subtle concerns.

Progress:
New start in FY88.

TITLE: Design and Analysis of Data Structures and Combinatorial Algorithms

PRINCIPAL INVESTIGATOR: Robert E. Tarjan

LOCATION: Department of Computer Science
Princeton University
Princeton, NJ 08544-0636

R&T No.: 4331667
CONTRACT No.: N0001487K0467

SCIENTIFIC OFFICER: Ralph Wachter

Technical Objectives:
The objective is to investigate efficient data structures, algorithmic methods and their applications. The specific objectives are: to devise specific data structures and methods tailored to the needs of various combinatorial problems, to perform theoretical and experimental studies of the new algorithms and their older competitors, and to elucidate general themes of design and analysis in the systematic construction of new algorithms.

Approach:
The approach to be followed by this research is to devise methods that can be applied to yield new and efficient algorithms for classical and newly defined problems in the areas of searching, sorting, networks, and computational geometry.

Progress:
A new heap data structure for implementing the classical shortest path algorithm have been developed in collaboration with R.Ahuja, K.Mehlhorn, and J.B.Orlin. These data structures are elegant, simple and theoretically efficient. There are many other significant research results to report on the topics of network flow, dynamic perfect hashing and Jordan sorting.

Ref.: R.Ahuja, K.Mehlhorn, J.Orlin and R.Tarjan, "Faster algorithms for the shortest path problem," TR-154-88, Department of Computer Science, Princeton University, Princeton, NJ, 1988.

Network Optimization Algorithms

Robert E. Tarjan

Department of Computer Science
Princeton University

Research Objectives

- To discover efficient computer algorithms for fundamental problems in network optimization

- To devise algorithmic tools, data structures, and analytic methods for the design and analysis of such algorithms

- To carry out empirical studies of such algorithms to learn what algorithms and implementation techniques are most practical

How fast can we solve

- Matching Problems?
- Maximum Flow Problems?
- Minimum-Cost Flow Problems?
- Variants and Extensions of these Problems?

Algorithms can be applied to

- Routing of messages, traffic, commodities
- Correlation of tracking data and targets
- Other combinatorial problems

Approaches

Algorithmic Techniques

- Successive Approximation
- Distance/Price Relabeling
- Parallel Processing
- (Randomization)

Data Structures

- Dynamic Trees
- Augmented Search Trees
- List Structures

Payoffs

- Maximum Flow: $O(nm \log\log U)$
- Minimum-Cost Flow: $O(nm \log(nC) \log\log U)$
- Assignment Problem: $O(\sqrt{n}\, m \log(nC))$
- Nonbipartite Assignment Problem:

$$O\left(\sqrt{n\alpha(n)\log n}\, m \log(nC)\right)$$

Primal Network Simplex for Maximum Flow:

$$O(nm \log n)$$

Generalized Primal Network Simplex for

Minimum-Cost Flow:

$$O\left(nm \log n \log(nC)\right)$$

Research Issues

What concepts are most important in algorithm design?

- Data structures
- Elegance

How can one exploit new concepts and technologies, such as randomization and parallelism?

- Development of alternative algorithms

What is required in a scientific approach to empirical analysis of algorithms?

- Challenging, realistic test data
- Consistency in coding

Karl Abrahamson
Department of Computer Science
Washington State Univ.
Pullman, WA 99164
509-335-2126
karl@cs2.wsu.edu

Chaderjit Bajaj
Department of CS
Purdue Univ.
West Lafayette, IN 47907
317-494-6003
bajaj@cs.purdue.edu

Peter Buneman
Computer and Information Science
Univ. of Pennsylvania
Philadelphia, PA 19104
215-898-7703
linc.cis.upenn.edu

Lowell Campbell
Department of Electrical Engineering
Univ. of Idaho
Moscow, ID 83843
208-885-6067

Robert Constable
Department of CS
Cornell Univ.
Ithaca, NY 14853
607-255-7316
rc@gvax.cs.cornell.edu

Michael Fellows
Department of CS
Univ. of Idaho
Moscow, ID 83843
208-885-7543
fellows@id.cs.bitnet

Christoph Hoffmann
CS Department
Purdue Research Foundation
West Lafayette, IN 47907
cmh@purdue.edu

Mark Hoover
CS Department
Univ. of New Mexico
Albuquerque, NM 87131
607-545-8366

Matt Kaufmann
Computational Logic Inc.
1717 W 6th St. Suite 290
Austin, TX 78703-4776
512-322-9951
kaufmann@cli.com

Roger King
Department of CS
Univ. of Colorado
Boulder, CO 80309
303-492-7398
roger@boulder.coloradoledu

Michael Langston
Department of CS
Pullman, WA 99164-1030
509-335-6486
langston@cs2.wsu.edu
(en route to Univ. of Tennessee)

Robert Paige
Courant Institute of Mathematical Sciences
New York Univ.
New York, NY 10003
212-~~[illegible]~~ 998-3000
paige@cims.nyu.edu

Joseph Pegna
Department of Mechanical Engineering
Univ. of California
Irvine, CA 92717
jpegna@orion.cf.uci.edu

Matthias Stallmann
Department of CS
North Carolina State Univ.
Raleigh, NC 27695-7003
919-737-2117
matt@csadm.ncsu.edu

Robert Tarjan
Department of CS
Princeton Univ.
Princeton, NJ 08544-0636
ret@notecnirp.princeton.edu

Tim Teitelbaum
Department of CS
Cornell Univ.
Ithaca, NY 14853
607-255-7573
tt@cs.cornell.edu

Ralph Wachter
Software Science
Office of Navel Research
800 North Quincy St
Arlington, VA 22217-5000
202-696-4304
wachter@itd.navy.nrl.mil